Automated examination timetabling with application to Stellenbosch University

Charles van Rooyen

Thesis presented in partial fulfilment of the requirements for the degree of
Master of Commerce (Operations Research)
in the Faculty of Ecomnomic and Management Sciences at Stellenbosch University

Supervisor: Dr Isabelle Nieuwoudt

December 2018

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2018

Abstract

The examination timetabling problem (ETP) consists of scheduling the examination papers of students in such a way that no student is required to write two or more examination papers at the same time in an examination period. It is well known that this problem is NP–complete, which makes it an interesting research topic for researcher. There exist many different variants of the ETP, and the one focused on in this project is the variant that can be applied to Stellenbosch University.

The purpose of this project is to find examination timetables that will spread most students' examination papers more or less equally over the full duration of the examination period. Stellenbosch University is used as case study. The primary objective of any algorithm is to satisfy the hard constraints provided by Stellenbosch University. For example, one hard constraint is that certain examination papers must be scheduled on fixed timeslots.

Graph colouring is used in a two–phase heuristic algorithm to obtain a feasible initial solution. In the first phase an examination timetable is sought where no student is required to write more than one examination paper during any timeslot. After the first phase, the number of examination papers scheduled during each timeslot is balanced in the second phase of the algorithm. This is to ensure that amongst others, enough lecture halls are available during each timeslot to accommodate all students.

After a feasible initial solution is obtained, hill climbing and the great deluge algorithm (GDA) are used to improve upon the equally spread of students' examination papers as much as possible over the entire examination period. Three moves are defined in this project to move from solution to solution in the solution space. The first move moves one module in the timetable to another timeslots, the second move swaps all of the modules in two timeslots and the third move is to swap two modules that are in different timeslots. To evaluate how well students' examination papers are spread over the entire examination period for each timetable, a newly derived cost function is used. The cost function strives to be fair towards all students. Parameter calibration is done on the parameters used in the cost function and the search algorithms.

The resulting timetables when using hill climbing and the GDA are compared, and it is found that the GDA outperforms hill climbing. Furthermore, the cost function used in this project is compared to the cost function of the 2nd International Timetabling Competition (ITC). Using Stellenbosch University's variant of the ETP, it is found that the cost function of this project outperforms the cost function used in the ITC.

Uittreksel

In die eksamenrooster–probleem (ERP) moet eksamenroosters sÃş ingedeel word dat geen student meer as een eksamenvraestel gelyktydig moet skryf nie. Hierdie probleem is NP–volledig en is al deeglik deur navorsers ondersoek. Daar is baie variante van die ERP, en in hierdie projek word daar gefokus op die variant wat op die Universiteit van Stellenbosch (US) van toepassing is.

Die doel van hierdie projek is om eksamenroosters op te stel wat studente se eksamenvraestelle so ver as moonlik eweredig oor die hele eskamenperiode versprei. Die US word as gevallestudie gebruik. Die primêre doelwit van enige algoritme is om aan die US se vaste vereistes vir 'n eksamenrooster te voldoen. Een van hierdie vereiste is, byvoorbeeld, dat sekere eksamenvraestelle op spesifieke tydgleuwe geskeduleer moet word.

Grafiekkleuring is gebruik in 'n heuristiek wat van twee fases gebruik maak om 'n aanvanklike eksamenrooster wat aan die US se vereistes voldoen, te kry. In die eerste fase word 'n eksamenrooster gesoek waarin daar van geen student verwag word om twee of meer eksamenvraestelle gelyktydig te skryf nie, en in die tweede fase word die hoeveelheid eksamenvraestelle per tydgleuf gebalanseer. Die tweede fase word byvoorbeeld benodig sodat daar ten alle tye genoeg eksamenlokale beskikbaar sal wees om alle studente te akkommodeer.

Nadat 'n aanvanklike toelaatbare oplossing verkry is, word *hill climbing* en *the great deluge algoritme* (GDA) gebruik om studente se opeenvolgende eksamenvraestelle so ver as moontlik uitmekaar te versprei. Drie eenvoudige skuiwe word in die algoritmes gebruik om sodoende deur die oplossingsruimte te beweeg. Die eerste skuif is om 'n eksamenvraestel te skuif na 'n ander tydgleuf, die tweede skuif is om al die eksamenvraestelle van twee tydgleuwe met mekaar om te ruil en die laatse skuif is om twee eksamenvraestelle wat in twee verskillende tydgleuwe is met mekaar om te ruil. Om 'n aanduiding te kry van hoe goed studente se eksamenrooster ingedeel is, is 'n koste–funksie bepaal. Die doelfunksie handhaaf regverdigheid ten opsigte van die hoeveelheid eksamenvraestelle wat per student geskryf moet word. Die parameters van die doelfunksie en die algoritmes word gekalibreer om sodoende goeie parameterwaardes te bepaal.

Die twee algoritmes word met mekaar vergelyk, en daar is bevind dat GDA beter vaar as die *hill climbing* algoritme. Verder word die koste–funksie van hierdie projek vergelyk met die koste–funksie van die 2de Internationale Roosterindelingskompetisie (ITC). Daar is bevind dat die koste–funksie van hierdie projek beter vertoon om die US se eksamenroosters in te deel as die koste–funksie van die ITC.

Acknowledgements

I would like to thank the following people who had a contribution towards this project:

- Dr Isabelle Nieuwoudt for all the coffee and cake, and most importantly for being an extraordinary supervisor.
- My mom for all her love, sacrifice and believing in me.
- Stellenbosch University for financial support and giving me the opportunity to go to Belgium on a semester exchange.
- Sarah van der Westhuizen for inspiring me to go on a semester exchange.
- Jaco and Nino for countless movies and food.
- Flora, NoÃľ, Kurt, Chesme, Phil, Pieter and Bryce for being amazing company in the post graduate lab.

Dedicated to

Everybody struggling with mental illness.

Contents

List of figures

List of tables

List of algorithms

CHAPTER 1

Introduction

Contents

"There are so many variations within the time–tabling schedule - we could write a book really!"

Anonymous university timetabler (1997)

Most universities have an examination period at the end of an academic year and/or semester. However, setting up good examination timetables are getting more difficult, because the number of students at universities keep on growing and they are also consistently being offered a wider variety of module choices [24]. Therefore, universities are constantly looking for better or different methods to set up an examination timetable.

1.1 Universities' examination timetables

The **examination timetabling problem (ETP)** consists of scheduling the examination papers of students in such a way that no student has *clashes* in his/her examination timetable [8]. A **clash-free timetable** would correspond to no student being expected to write two or more of his/her examination papers during the same timeslot. It is well known that this problem is NP-complete [13], which makes it also an interesting research topic for researchers.

There exist many variants of the ETP, since each university has its own requirements for the examination period [12]. These variants usually have hard and soft constraints derived from the specific university's needs. **Hard constraints** must be met in order to have a feasible solution, whereas **soft constraints** are more flexible. Staying within the boundaries of the soft constraints should normally yield solutions better fit for the university's needs. To list all existing constraints of all the variants of the ETP would be difficult, since every university's needs could potentially be another variant of the problem [12].

The two constraints that are almost always present are that no student may have any clashes in his/her timetable, and that there must be enough room in the available venues to accommodate all students during the examination period [39]. These two constraints are mostly used as hard constraints. Some universities also require that examination papers of the same length should be written in the same timeslot as to cause fewer disturbances, because there would not be a lot of students leaving the examination venue while other students are still writing. Minimising consecutive examination papers of students is a soft constraint that most universities use, as this could potentially improve the students' performance during the examination period [39]. Another soft constraint focused on improving student performance, would be to try and maximise the time a student has to study for his/her examination papers.

For some universities it is important that certain examination papers must be written before certain other examination papers may be scheduled. Ensuring that lecturers are available for their examination paper's timeslot is important to some universities [12]. In this constraint, a lecturer should not have examination papers for two or more modules that he/she taught, scheduled during the same timeslot. For some examination papers, the use of special equipment or computers, could be required, and thus the timetable must be scheduled in a way that the necessary venues are available at the given timeslot. Another constraint that some universities use is that modules with a large number of students should be written early in the examination period as to give the lecturers enough time to mark the examination papers before the deadline. Lecturers could also have special preferences as to when their examination papers should be scheduled, based on when they are available. These are just a few of the constraints that could be in the variants of the ETP [13].

1.2 Timetabling at Stellenbosch University

The academic year at Stellenbosch University corresponds with a calender year, meaning that the first semester usually start in the beginning of February. There are two examination periods per year at Stellenbosch University, one at the end of each semester. Examination periods usually start on a Tuesday and examination papers are scheduled for every day of the week, except for Sundays. Furthermore, there are two timeslots per day, namely one morning slot and one afternoon slot. Examination periods at Stellenbosch University consists of two parts. The first part of the examination period is to provide students with a first opportunity to write their examination papers. The duration of the first opportunities last for a total of 18 examination days during the first semester's examination period and last for 20 examination days during the second semester's examination period. After the first opportunity, a second opportunity period starts, which gives students another chance to complete their modules in case they failed the first examination paper or they were unable to write the first opportunity's paper due to illness. The second opportunity period usually lasts for 15 days.

To gather more detail on the methods which are currently being used to obtain the university's timetables, a meeting was held with S. Franken [25], Stellenbosch University's head of timetables. During this meeting the specific requirements for Stellenbosch University's examination timetables, were also determined.

Currently, the head of timetables creates the university's examination timetables at the end of the year prior to the start of the new academic year by using three steps. The first step is to use software provided by *Scientia* to get an initial timetable [25]. *Scientia* is an examination timetabling software for higher education institutions that minimise back-to-back examination opportunities of students [41]. An initial timetable for Stellenbosch University is determined with

the previous year's list of student enrolments (*i.e.* each student with his/her module choices for the year) is used as input for *Scientia*. After the input is uploaded to *Scientia*, the software takes about a minute to give a timetable as output [25].

There are, unfortunately, not enough features built into *Scientia* to take the needs of the lecturers and the university into account when it comes to Stellenbosch University's examination timetables [25]. Thus, after the initial timetable is obtained, the head of timetables must manually move modules in the initial timetable to better suit lecturers' needs. These manual adaptations are the second step in the process of obtaining an examination timetable for Stellenbosch University. This is a cumbersome process and takes about 2 months to complete.

The hard constraints that lecturers provide the head of timetables include that some specific examination papers must be scheduled in timeslots later than some other examination papers. Another hard constraint is that some examination papers must be scheduled for a certain date and/or timeslot. The final hard constraint is that certain examination papers may not be written on the same day as specific other examination papers [25].

There are also soft constraints that the head of timetable must try to satisfy by manually adapting the initial examination timetable. The requirement that a certain examination paper should be scheduled during a certain week is the first soft constraint. Other soft constraints include that certain examination papers should not be scheduled during certain weeks or that some examination papers should not be scheduled in the last three days of the examination period. Some of the requests from lecturers also include that the examination papers of a module that the lecturer teach, should be scheduled in a morning timeslot or that certain examination papers should be on a certain day of the week. These requests can also include that certain examination papers should not be scheduled on or before a certain date or that the examination paper should be scheduled as early as possible in the examination period. The last request that the head of timetables receives from lecturers is for certain examination papers to be scheduled earlier in the examination period than some other examination papers [25].

The third step of setting up an examination timetable at Stellenbosch University is to assign venues to the different timeslots. Space in the venues is not usually a problem at Stellenbosch University, since there are more than enough lecture halls available to use. *Scientia* does, however, take into account the number of venues available versus the number of students required to write their examination papers during a certain timeslot when setting up the initial timetable [25]. Even though space is not usually a problem at the university, this constraint cannot be completely ignored. If this constraint is ignored during the second step of setting up the examination timetable it could, for instance, cause a large number of examination papers being assigned to one timeslot, resulting in a lack of available space for this large number of venues needed.

1.3 Problem description and objectives

Reflecting on the information that is given by the head of timetables at Stellenbosh University, at least three problems are identified regarding the current process of setting up examination timetables for the university. The first of these is the time that it takes a person to set up the timetable. To get the initial timetable using *Scientia* is fast, but manually swapping examination papers afterwards is not. This is a frustrating and time consuming 2–month back-and-forth communication between the lecturers and the head of timetables.

The second and third problems arise because of the manual swapping of examination papers

in the initial timetable. The initial timetable was obtained via *Scientia* and thus consecutive examination papers of students were minimised. If examination papers are manually swapped within the initial timetable, it would most likely undo the effort via *Scientia* of trying to minimise consecutive examination papers of students. This is the second problem that is identified. *Scientia* also takes into account the number of venues available to accommodate students during the examination period. The number of venues available could also potentially cause problems after the swapping of examination papers, and as such becomes the third problem that is identified. Even though space is not usually a problem at Stellenbosch University, it could be an issue if the head of timetables does not manually inspect the timetable to make sure that there is enough room available to accommodate all students after she made a swap within the timetable.

It will therefore be desirable to solve all three steps described in §1.2 simultaneously in order to overcome the problems mentioned above. One thus want to define, formulate and solve a variant of the ETP that best suit the requirements of Stellenbosch University. In order to define the specific variant of the ETP used in this project, the hard constraints are given first. Thus, a feasible solution in the specific variant of the ETP focused on in this project is a examination timetable in which

1. no student is required to write more than one examination paper in any timeslot,
2. examination papers that must be written in fixed timeslots are scheduled in their predefined timeslots,
3. examination papers that must be written simultaneously are scheduled in the same timeslot, and
4. the number of examination papers scheduled per timeslot must be approximately equal.

Requirement 1 in the list above is the universal hard constraint used in almost all variants of the ETP. The second constraint, Requirement 2, is included since lecturers at Stellenbosch University could require particular examination papers to be written on fixed timeslots. The reason for these fixed timeslots could be because the examination papers must be externally moderated by a certain date, the lecturers are only available to invigilate their modules' examination papers during those specific timeslots or simply because these timeslots are convenient for the lecturers.

Since in some cases at Stellenbosch University different modules have the same examination paper, Requirement 3 serves as the third constraint. If some of these students write their examination paper during an earlier timeslot, these students are able to give the other students valuable information on the examination paper before they themselves will write the paper. This situation of some modules having the same examination paper typically arises when students from departments in different faculties are enrolled for the same module, but the module has a different name in each department.

Finally, to ensure that there is enough room available in the examination venues to accommodate all of the students during each timeslot, Requirement 4 was added. It is assumed that if all the examination papers per timeslot is approximately equal, then all of the students will fit into the examination venues. This assumption should be accurate, since Stellenbosch University has plenty of venues to use during the examination period.

The main soft constraint that is focused on in Stellenbosch University's variant of the ETP, is to spread each students' examination papers over the whole examination period as evenly as possible. The number of examination papers that each student must write is taken into consideration to make the optimisation of the soft constraint fair towards all students. Since

spreading each student's papers over the whole period is one of the major aims of this project, this soft constraint is to be used in the objective function of the search algorithms, while the hard constraints must be satisfied to get a feasible solution. There are some other soft constraints that the head of timetables receives from lecturers that are not incorporated in the variant focused on in this project. Methods of dealing with these soft constraints is discussed in §5.2.

The approach to solve the above variant of the ETP for Stellenbosch University, can be summarised in the following objectives:

I. Set up an usable data structure for representing the timetable at Stellenbosch University.

II. Identify a method to obtain a feasible initial solution in which all the hard constraints are satisfied.

III. Derive an objective function to evaluate how well students' examination papers are spread over the entire examination period.

IV. Implement two search algorithms to improve the initial solution.

V. Compare the results obtained from the two algorithms.

VI. Make suggestions on further improvements that can be made to the algorithms.

1.4 Thesis layout

In Chapter 2 literature is discussed on examination timetabling, with the focus on graph colouring and local search algorithms. The selection of the solution approach is discussed in §2.2, and graph theory terminology necessary for this project is given in §2.3. The methodology is discussed in Chapter 3, starting in §3.1 with constructing a timetabling graph which addresses Objective I. Deriving a method to achieve Objective II of finding a feasible initial solutions is done in §3.2. Objective III, *i.e.* deriving a function to evaluate how well students' examination papers are spread out over the entire examination is done in §3.4, while in §3.3 the moves implemented in the search algorithms, discussed in §3.5, are explained. Thus, these two sections address Objective IV. The application of the proposed algorithms on Stellenbosch University's datasets is discussed in Chapter 4. Parameter calibration is done in §4.2, after the data manipulation for Stellenbosch University's timetabling graph was explained in §4.1. The two algorithms are compared in §4.3.2 and thus Objective V is the topic in this section. The objective function used in this project is also compared to a popular objective function used in literature. Finally, recommendations on how the search algorithms can be improved for future use is discussed in Chapter 5 (Objective VI).

CHAPTER 2

Literature

Contents

"The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them."

Sir William Bragg (1915)

Several approaches for solving the different variants of the ETP have been studied throughout the years. These approaches include, just to name a few, the use of various metaheuristics, heuristic algorithms, graph colouring methods, integer programming, neural networks, and constraint programming. It would be possible to write a complete textbook on all the research that have been done on the topic of the ETP. It will thus not be attainable to give a complete literature review here and those approaches that will not be considered in this project will be omitted entirely. For example, it is well known that an integer programming (IP) problem is NP-hard [40] and the amount of effort required to formulate the constraints as well as the fact that computational difficulties arise in large, real world combinatorial problems, makes this method unpopular for solving the ETP [20]. Furthermore, in most cases where integer programming is used, it is only used to get an initial solution for the ETP [22]. This method is much more suitable for solving smaller problems, like the school timetabling problem [44]. Stellenbosch University's variant of the ETP is simply be too large and have too many constraints to be efficiently solved via integer programming. It is for this reason that integer programming will not be considered in this study.

Therefore, only the relevant ideas from literature that was used in this project, will be summarised in this chapter, starting in §2.1 with some articles that provided valuable insight about the ETP. Next, the reason for choosing the specific solution approach for this project is explained in §2.2. Lastly, in §2.3, the relevant graph theory definitions needed in order to understand the methods used in this project are given.

2.1 Examination timetabling through the years

In one of the first papers where the ETP is represented with graph theory, Welsh and Powell [45] described in 1967 a method to solve the ETP in such a way that no students have any clashes, and the number of timeslots used in the examination period is minimised. Their method makes use of proper coloring to identify the upper bound for the chromatic number of the graph. In practice, their method is only useful to find the minimum number of timeslots needed in an examination period so that no student has clashes. This is not very practical, as students could potentially write all of their examination papers in consecutive timeslots. Another problem is the number of examination papers per timeslot could vary a lot. Using this method, one could potentially schedule 100 examination papers in a single timeslot and only one examination paper in another timeslot. Welsh and Powell's method will not be considered in this project, since the variation they use differs a lot from the variant focused on in this project.

A method that was used to schedule examination timetables for Manchester University was described by Wood [46] in 1968. His method involved the use of a conflict matrix to see whether or not certain examination papers could be scheduled in the same timeslot. Wood's method did not allow any students to have clashes, but students could potentially have more than one examination paper per day. This is because his method only minimises the number of papers that a student writes in a 24 hour time period. Wood was able to set up an examination timetable that lasted for 30 days, where most of the examination papers were scheduled during the first 20 days as to give lecturers enough time to mark the papers of the modules they teach. Wood's method had fewer clashes than manually built examination timetables at Manchester University, and the examination period was also shorter than the manually built timetables [46]. Wood's method will not be considered in totality in this project, since he does not space out students' examination papers over the whole examination period. Minimising the number of examination papers that a student must write within 24 hours could still make the student write all of his/her papers in one week, which does not solve the problem of trying to space out a student's papers evenly over the whole examination period.

Brelaz [7] presented a heuristic method in 1979 to colour the vertices of a graph in order to solve the ETP. His heuristic relied on the comparison of the degrees of the vertices in the graph. He would first colour the vertex with the highest degree, introducing the first colour. After that he colours the vertex u with the maximal saturation degree[1]. If there is more than one vertex of maximal saturation degree, any one of them is chosen to be u. If vertex u can be coloured with a colour that had already been introduced and without having the same colour in its neighbourhood, u is coloured with that colour. Otherwise a new colour is introduced. Brelaz describes a good heuristic to solve a simple version of the ETP [7], but his method is not practical. It does not provide a way to ensure that lecturers' and students' needs are met, and thus his heuristic will not be considered in this project. However, the idea of using saturation degrees to colour vertices will be experimented with when a method to find an initial solution in this project, is sought.

In 1979, Leighton [28] described a method using graph colouring to solve large scheduling problems. He found that, in general, it is a good idea to colour vertices with a large degree first, followed by the vertices with smaller degrees. Leighton does not describe a way to incorporate any hard or soft constraints, so his method will not be used in this project. The idea behind colouring vertices with the highest degree first will also be considered in a method of finding an initial solution for the ETP used in this project.

[1]The saturation degree of a vertex is an indication of the number of colours that may *not* be used to colour that vertex, since this is the number of colours already in its neighbourhood.

Mehta [34] came to the conclusion in 1981 that the objective of examination timetabling is not always simply to find the minimum number of timeslots for an examination timetable without having any clashes in the timetable. He points out that a university can ask for the examination period to be, for instance, 10 days long. If one uses a method to determine the least number of days for the examination period and the answer obtained is larger than 10 days, the method will not work to solve the particular university's problem. Mehta describes a simple graph colouring heuristic to solve this problem while keeping the number of clashes to a minimum [34]. This heuristic can not be considered in this project, since clashes will not be allowed in Stellenbosch University's examination timetable.

Eight existing methods to colour vertices in such a way as to minimise the number of colours used so that no two adjacent vertices have the same colour, were described by Carter et al. [17] in 1986. About 10 years later, in 1996, Carter et al. [19] used their methods to develop 40 different strategies to solve the ETP and compared them to one another. The best strategy they developed was to colour the vertices within the largest clique first. Within the largest clique they would also colour the vertices with the largest saturation degree first. They also incorporated backtracking to improve their heuristic. They did not use any soft constraints when running the tests, which makes their strategies less useful for this project. They improved on their initialising method in 2001 by incorporating the use of cliques for initialisation [18].

In 1997, Bullnheimer [9] described a simulated annealing approach to solving the ETP on a small scale. His algorithm tries to maximise the time students have to study for their examination papers. He makes use of four different moves to generate his neighbourhoods. The first move is to swap the timeslots of any two distinct modules within the current timetable. The second move is to choose a random sequence within the current timetable and then move the sequence to a different spot. For example, suppose the examination papers a, b, c, d, e, and f in the current timetable are scheduled in the order $b-c-a-d-f-e$. The algorithm then randomly chooses a sequence, say $c-a-d$, and places it in a different spot, also randomly chosen. A new timetable might be, for instance, $b-f-c-a-d-e$. The third move is to choose a random sequence and then invert the sequence within the timetable. Using the previous example, *i.e.* the sequence $c-a-d$ in the timetable $b-c-a-d-f-e$, the new timetable would be $b-d-a-c-f-e$. The last move is to randomly rearrange sequences of examination papers within the current timetable. He found that maximising studying time seems to lead to more fairness in the timetables [9].

Thompson et al. [43] described a 2–phase method in 1998 to solve the ETP by making use of simulated annealing. The first phase is to find a feasible answer according to the hard constraints. In the second phase, the metaheuristic optimises the solution by taking the soft constraints into account while still satisfying hard constraints. The only soft constraint that they experimented with was to minimise consecutive examination papers written by students. They used two moves to define two different ways of generating a neighbourhood. The first move was very simple since one just simply change the timeslot of one examination paper within the timetable by changing the colour of the corresponding single vertex within the graph. The second move consists of using any pair of colours to generate a Kempe chain[2], and then swapping the colours of the vertices within the Kempe chain. They found that although it takes more computational time, using Kempe cains in their moves improved their solution more than just simply changing the colour of one vertex in a move [43].

In 1998, a heuristically guided search to solve the ETP, was described by Burke et al. [14]. They experimented with backtracking combined with sequencing strategies, similar to the approach used by Carter et al. [19] in 1996. The big difference in their strategy is that they use an

[2]The Kempe chains used in Thompson et al. [43] is based on the colour chains developed by AB Kempe in his attempted proof of the Four-colour Theorem.

objective function to determine the penalty of a module being placed in a certain timeslot, and then placing the module in the timeslot that results in the lowest penalty. The objective function took three factors into account. The first factor, and by far the highest weighed factor in the objective function, is the number of clashes that will follow after the examination paper is scheduled in a certain timeslot. The second factor involves looking at how many students would have to write more than one examination paper on the same day, but in different timeslots. In the last factor, the number of students that have to write examination papers on consecutive days after the examination paper is schedule in that timeslot, is considered. They found that this method of guided heuristic searches can improve on simple heuristic search models [14].

A 3–phase method to solve the ETP was described in 2002 by Merlot et al. [35]. In the first phase, an initial solution is found by making use of constraint programming. The second phase consists of improving the solution by using simulated annealing, followed by the third phase that uses hill climbing to further improve the solution. For their simulated annealing algorithm they used a normal geometric cooling function together with Kempe chains to generate their neighbourhoods. They tested their algorithm on 14 different data sets and found that simulated annealing on its own performed poorly, since testing for feasibility and finding feasible solutions took a lot of computational power. Incorporating hill climbing in the third phase improved their solutions significantly. Their objective function just penalise consecutive examination papers being written by a student [35], which is not useful for spacing out the students' examination papers evenly.

Duong et al. [24] described a 2–phase method in 2004 to solve the ETP. The first phase makes use of constraint programming to get an initial solution that satisfies the hard constraints. In the second phase they use simulated annealing, together with their soft constraints, to optimise their objective function. They use the same Kempe chains as Thompson et al. [43] to generate their neighbourhoods. Their objective function is written in terms of the total number of students n_{ij} who enrol for both modules m_i and m_j [24]. For high values of n_{ij}, the algorithm would then try and space out modules m_i and m_j as far as possible within the examination timetable. The objective function will unfortunately not space out students' examination papers evenly, which makes it undesirable to consider for this project. To demonstrate, let there be a student A who enrolled for modules m_1, m_2, m_3 and m_4, while student B enrolled for modules m_1, m_2, m_5, and m_6. Hence, $n_{12} = 2$ and all other $n_{ij} = 1$. The objective function will thus force examination papers m_1 and m_2 to be written as far apart as possible, but since all other $n_{ij} = 1$ it would not matter where the other modules are placed within the timetable. This means that a timetable with the sequence m_1, m_3, m_4, m_5, m_6, m_2 will have the same objective function as a timetable with sequence m_1, m_3, m_5, m_4, m_6, m_2. It is easy to observe that the second sequence would be much better in terms of spacing out students' examination papers. In the first sequence both student A and student B will have to write three of their papers in a row, whereas in the second sequence they would only be required to write two papers in a row.

In 2004, Burke and Newall [15] described an adaptive heuristic that provided an alternative to existing forms of backtracking. Their idea was to schedule examination papers based on how 'difficult' the modules were to schedule. Examination papers of modules taken by a wide variety of students would, for example, be considered as difficult to schedule, since this examination paper could potentially clash with a lot of other modules' examination papers. Their heuristic sorts the examination papers according to difficulty, whereafter the most difficult examination papers to schedule, are scheduled first. If the heuristic reaches a point where it is impossible to schedule an examination paper without causing a clash in the timetable, the heuristic would restart. When the heuristic restarts, it adjusts the 'difficulty' of scheduling the examination papers according to what happened in the previous run of the heuristic. An examination paper

that could not be scheduled will have an increased difficulty after the restart, so the examination paper will be scheduled earlier in the next run of the heuristic. They found that this method requires less computational power than using the previous implementations of backtracking [15].

Burke et al. [11] described an adaptation of the great deluge algorithm [23] to solve the ETP in 2004. The great deluge algorithm is very similar to simulated annealing, where the difference is in the way that these algorithms will accept a solution with a weaker value than the current solution. The great deluge algorithm starts off with an acceptance level θ. Whenever a new solution is accepted as the current solution, the value θ decreases by making use of a function $D(\theta)$. Moving from the current solution to a new solution, the new solution will only be accepted if the value of the objective function is lower than θ. Burke et al.'s algorithm only make use of one move to generate their neighbourhood, namely to move one examination paper to another timeslot within the current timetable. The use of different neighbourhoods and initial solutions were not explored in their work. They found that the great deluge algorithm is a good alternative to simulated annealing, since it is much easier to determine good paramaters for the great deluge algorithm than it is to obtain good paramaters for simulated annealing [11].

During 2007, Carrington et al. [16] developed a method to solve the ETP by means of colouring a weighted graph. The aspect that makes this method different than most colouring heuristics, is the fact that they use weights on the edges to determine the colouring of the vertices. They do not make use of proper colouring at all. Instead, the weight on the edges would be a number that reflects how desirable it would be to colour the two vertices that joins the particular edge, with the same colour. These weights take into account the number of students that have to write both examination papers represented by the vertices, together with a factor that states how desirable it would be to have those two examination papers be scheduled on the same timeslot [16]. Their method allows students to have clashes, but it keeps the number of clashes to a minimum. Their method will not be considered in this project, since the examination timetable of Stellenbosch University must be free of clashes. The method by Carrington et al. [16] also do not provide for a way to incorporate different soft and hard constraints, which is needed in this project.

A simulated annealing approach to solving the ETP was described by Battistutta et al. [3] in 2014. They allow for infeasibility, but penalise it heavily in the objective function. For an initial solution, they start with a randomised timetable, which is almost always infeasible. They generate their neighbourhoods by using two simple moves. The first move is to move one examination paper to another spot in the timetable, while the second move consists of swapping two examination papers in different timeslots within the timetable with one another. On each iteration, they choose which move to use by using probability r for the first move and probability $(1 - r)$ for the second move. Their algorithm starts off with a high initial temperature and their cooling scheme initially cools the temperature very fast. They found that using a straightforward searching method, together with a cooling function that accurately calibrates it's parameters after each iteration, can give good solutions to the ETP [3]. This algorithm was developed for the variant of the ETP as formulated in the 2nd International Timetabling Competition [30]. The variant of the ETP that they use is not the same as the variant of Stellenbosch University, so this algorithm as a whole will not be considered during this project. Furthermore, allowing any infeasiblility in the solution is not wanted in this project.

2.2 On the selection of the solution approach

As mentioned, there are various solution approaches in literature. For this project, it was decided to use metaheuristics to find a graph colouring solution to the specific variant of the

ETP considered.

Formulating the ETP in the form of a graph colouring problem has been a popular way of modeling the ETP since the late 1960's. Another popular way of modeling the ETP as discussed in §2.1, is to use a two dimensional conflict matrix C to observe which examination papers may not be scheduled in the same timeslot. Each row i and column j of the conflict matrix corresponds with an examination paper, and if element $c_{ij} = 1$ it means that paper i and j may not be scheduled in the same timeslot. This conflict matrix and the adjacency matrix of a graph contains exactly the same information, though. The manner in which the problem is presented is therefore a matter of preference. In this project the ETP will be modelled and solved via graph colouring.

In recent years, metaheuristics have become more and more popular in solving discrete optimisation problems with a large solution space, since the speed and memory of computers have been improving immensely. Thus, metaheuristics are able to find good approximate solutions to NP-complete problems like the ETP [5]. It is for this reason that metaheuristics is incorporated in this project.

There exist many metaheuristics in literature that are proven to work relatively well when applied to the ETP. The two metaheuristics that are most commonly used to solve the ETP is simulated annealing and the great deluge algorithm. It was decided to focus on the great deluge algorithm in this project, since the parameters of the great deluge algorithm are easier to calibrate than the parameters of simulated annealing, while still obtaining similar results [11].

2.3 Graph theory terminology

Graph colouring is a useful tool when modeling scheduling problems [29]. Before the graph that will be used to represent the variant of the ETP used in this project can be formulated, a few terms need to be defined.

A **simple graph** $G(V, E)$ consists of a finite, nonempty set $V(G)$ of elements, called **vertices**, together with a (possibly empty) set $E(G)$ of elements, called **edges**, which are 2–element subsets of $V(G)$ such that uv (or vu) denotes the edge between u and v, where $u, v \in V(G)$. The total number of distinct vertices in a graph G is called the **order** of G, denoted by $p(G)$, and the total number of distinct edges in G is called the **size** $q(G)$ of G. The edge $e = uv \in E(G)$ **joins** vertex u and v with each other, and vertex u and v are said to be **adjacent** vertices. A popular way of representing a graph is graphically with points and lines where the points represent the vertices and there exists a line between two vertices if the two vertices are adjacent.

The **neighbourhood** of a vertex u in a graph G, denoted by $N_G(u)$, is a set containing all the vertices adjacent to vertex u and any vertex v in this set $N_G(u)$ is called a **neighbour** of vertex u. The **degree** $deg_G(u)$ of a vertex u in a graph G is defined by the number of distinct vertices adjacent to u, *i.e.* $deg_G(u) = |N_G(u)|$. Any graph H in which $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ is called a **subgraph** of graph G [6]. In a graph G, a **clique** is a subgraph of G in which every pair of vertices are adjacent to one another [42].

A **colouring** of a graph G consists of assigning colours to the vertices of G, in such a way that every vertex is assigned exactly one colour. If a colouring of a graph G consists of x colours, $V(G)$ can be partitioned into x sets, called **colour classes**, such that each vertex of a particular colour class has the same colour. A graph colouring in which no adjacent vertices have the same colour is called a **proper colouring** [36]. An **equitable colouring** of a graph G is a proper colouring of G in which the cardinality of any colour class differ by at most one from

the cardinality of any other colour class [27]. A **k–bounded colouring** of a graph G is a proper colouring of a graph G in which no colour class has a cardinality larger than k [26]. The **saturation degree** $\varrho(u)$ of a vertex u in a coloured graph G, is defined as the number of distinctly coloured vertices adjacent to u [4].

CHAPTER 3

Methodology

Contents

"Every once in a while, a new technology, an old problem, and a big idea turn into an innovation."

Dean Kamen (2016)

In this chapter the approach to solving Stellenbosch University's variant of the ETP is discussed. The process of presenting this specific variant of the ETP in the form of a graph colouring problem is explained in §3.1. The solution approach that will be followed, is to search from one solution in the solution space to another solution in the solution space, in an attempt to find a better timetable. To start the search process, an initial solution is needed. The approach to obtaining an initial solution is explained in §3.2. Different moves used in the search algorithms to move from one solution in the solution space to the next, are described in §3.3. In order to decide whether one solution in the solution space is better than another one, an objective function is needed. The objective function used in this project is a cost function, derived in §3.4. Finally, the two different search algorithms that will be used, as well as how their parameters are to be calibrated, are explained in §3.5.

3.1 The timetabling graph

The data structure used in this project will be a graph as defined in §2.3 in order to use graph colouring as solution approach. More particularly, let a **timetabling graph** be a graph in which the vertices represent the different examination papers that must be scheduled for a specific examination period, and an edge joins a pair of vertices if the examination papers represented by these two vertices may not be written simultaneously. Examination papers that may not be written during the same timeslot is mostly due to the existence of at least one student that is required to write both papers. There are two ways to obtain all the edges of a particular timetabling graph. The first is to use all possible module combinations that are allowed at the specific university to obtain a timetabling graph. By using this, all examination papers that may not be written during the same timeslot are identified. All pairs of vertices that represent the examination papers of two modules for which the same student *may* enrol, are joined by an edge. However, if one takes all possible combinations of module choices into account, it is possible that there exist several pairs of adjacent vertices in the timetabling graph for which there is no student that is actually enrolled for both modules that correspond to these vertices. This would put unnecessary constraints on the problem and could limit possibilities of optimisation. It is for this reason that it was decided to use the second method from literature to obtain the edges of the timetabling graph.

The second method of obtaining the edges of the timetabling graph avoid the complication mentioned above by using the list of actual student enrolments to identify the edges. Most universities use the list of student enrolments to get the edges of the timetabling graph by calculating the neighbourhood of each vertex [39]. The vertices that are representing the examination papers of a student in the list of enrolments are simply added to each other's neighbourhoods. By doing this for all students, all of the edges can be obtained. This method is illustrated in Example 3.1.1.

Example 3.1.1. Suppose there are five different examination papers that must be scheduled. Let Table 3.1 contain the list of students and the examination papers that each student should write.

Student ID	List of examination papers
0	Mathematics, English, Science
1	English, Biology
2	Science, Accounting, Biology

TABLE 3.1: *List of student enrolments for Example 3.1.1.*

This means that there will be five vertices representing each of these examination papers in the timetabling graph as seen in Figure 3.1(a). Considering the student with *Student ID* $= 0$ in Table 3.1, it is clear that Mathematics, English and Science may not be written during the same timeslot. Thus, all three vertices representing these papers are added to each other's neighbourhoods and the edges between those vertices are included in the graph. The resulting graph can be seen in Figure 3.1(b). The next student with *Student ID* $= 1$ in Table 3.1, is enrolled for both English and Biology and thus these two modules may also not be written during the same timeslot. The vertices representing these two examination papers are added to each other's neighbourhoods and the edge between them is added to the graph, as seen in Figure 3.1(c). The final student with *Student ID* $= 2$ in Table 3.1 suggests that Science, Accounting and Biology may not be written during the same timeslot. The vertices are added to each other's neighbour-

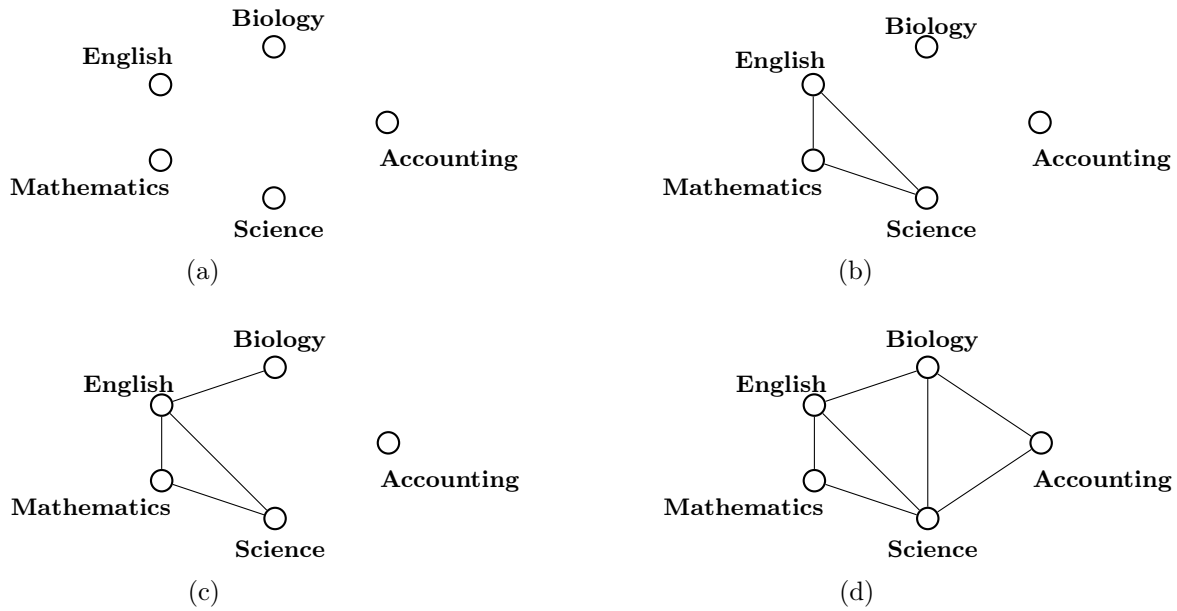


FIGURE 3.1: *Graphs used in Example 3.1.1 to obtain a timetabling graph: (a) the initial graph contains the modules (vertices) only and (b) [(c) and (d), respectively] the resulting graph after the modules for which student with Student ID* $= 0$ *[Student ID* $= 1$ *and Student ID* $= 2$*, respectively] are enrolled, are included in the graph.*

hoods, and the edges between them are added to the graph. The final graph can be seen in Figure 3.1(d). ■

After the timetabling graph was obtained, an examination timetable may be generated by finding a proper colouring of the timetabling graph. Thus, to obtain a solution to the ETP let every colour represent a distinct timeslot. If a vertex has a certain colour, the examination paper represented by the vertex gets scheduled during the timeslot represented by the colour of the vertex [21]. Note that a *proper colouring* of a timetabling graph will produce an examination timetable in which no student is required to write more than one examination paper at a time, since no adjacent vertices have the same colour.

However, before a colouring is sought, based on the specific variant of the ETP used, the general examination timetabling graph as described above may have to be adapted to incorporate some of the specific requirements for the specific variant of the ETP used. The requirements for a feasible solution for the variant used in this project, was listed in §1.3. These requirements may be translated into a graph colouring in which

1. no pair of adjacent vertices have the same colour,
2. vertices with predefined colours must have those colours,
3. some predefined vertices must have the same colour, and
4. the cardinality of the colour classes must be approximately equal.

Requirement 1 means that the solution must be a proper colouring, so that a clash-free examination timetable may be obtained. Thus, Requirement 1 must be satisfied during the colouring process. The same applies for Requirement 4, which is to ensure that the number of examination

papers scheduled during each timeslot is more or less the same. The vertices in Requirement 2 would correspond with the examination papers that must be written on fixed timeslots, where the predefined colours correspond to these timeslots. Hence, this requirement can be satisfied by colouring these vertices with the corresponding colours first.

If some examination papers must be scheduled during the same timeslot, it means that the vertices representing these examination papers must have the same colour, which leads to Requirement 3 in the list above. This requirement can be simplified by adjusting the timetabling graph in the sense that vertices that are required to have the same colour are *merged*. This is done by replacing the vertices that must have the same colour with one vertex, where the new vertex is adjacent to all of the neighbours of the vertices merged into it. This new vertex will thus represent more than one examination paper that needs to be scheduled, but it will ensure that the papers are scheduled during the same timeslot since this vertex will only have one colour. It is clear that this merging is done without the loss of generality, since the new vertex will still be connected to all the neighbours of the old vertices. Therefore, this change cannot cause any clashes within the timetable as long as a proper colouring is obtained. The concept of merging vertices and then colour the vertices having fixed colours first, is illustrated in Example 3.1.2.

Example 3.1.2. Suppose the same information is given as in Example 3.1.1. Furthermore, let Table 3.2 contain the information on the examination papers, and the timeslot represented by each colour is given in Table 3.3.

Examination paper	Fixed date	On the same time
Mathematics	Timeslot 2	Accounting
English		
Science		
Biology		
Accounting		Mathematics

TABLE 3.2: *Information on the examination papers for Example 3.1.2.*

Timeslot	Colour
1	Blue
2	Red
3	Green
4	Yellow

TABLE 3.3: *The timeslots with the colour representing each timeslot for Example 3.1.2.*

The timetabling graph was obtained in Figure 3.1(d). From the third column in Table 3.2, it can be seen that Mathematics and Accounting must be written on the same day. The vertices representing these two examination papers are merged, forming the graph in Figure 3.2. Note that the new merged vertex is connected to all of the neighbours of both Mathematics and Accounting. No more papers need to be written on the same timeslot, so the graph in Figure 3.2 is the final adjusted timetabling graph.

Before any colouring algorithm is applied to the adjusted timetabling graph, any vertex that must have a predefined colour must be coloured with this colour. From Column 2 in Table 3.2, the vertex that represents Mathematics must be assigned the colour representing Timeslot 2. The colour of the vertex will thus be red, as indicated in Table 3.3. The resulting graph can be seen in Figure 3.3. Note that the merged vertex now forbids the vertex that represents

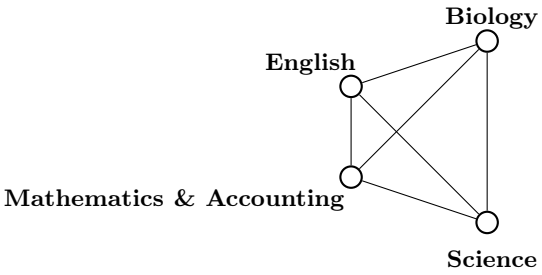


FIGURE 3.2: *The final adjusted timetabling graph after the vertices representing Mathematics and Accounting have been merged in Example 3.1.2.*

Biology to be coloured red, as they are adjacent. This would not have been the case if only Mathematics was coloured with red in Figure 3.1(d) before the merging happened. But the fact that Accounting is part of the merged vertex in Figure 3.3 causes red not to be available to Biology, since Accounting and Biology were adjacent in Figure 3.1(d). If the merged vertex was to be split up again, as can be seen in Figure 3.4, no information is lost. Biology may still not be red, for example. ■

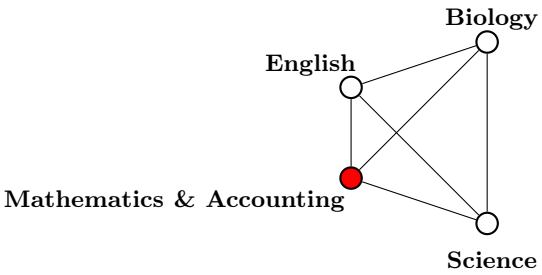


FIGURE 3.3: *The resulting graph after the merged vertex in the graph of Figure 3.2 is coloured red, because red is Mathematics' fixed predefined colour according to Table 3.2 in combination with Table 3.3.*

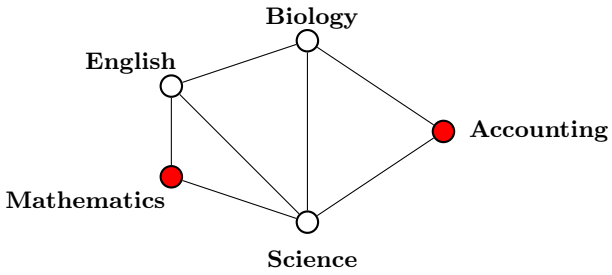


FIGURE 3.4: *The resulting graph after one would have split the coloured merged vertex in the graph of Figure 3.3.*

3.2 Initial solution

Finding an initial feasible solution for the ETP is an important part of the **search algorithms** that will be used to search for better solutions, and will be discussed in §3.5. It is known that most versions of the timetabling problem result in a large combinatorial problem [32], meaning that if one would start with an infeasible initial solution, it could be difficult to obtain feasibility from this solution again. Thus, it might be easier to start with a feasible solution. In terms

of our graph colouring problem, a feasible initial solution would be a proper colouring of the timetabling graph in which vertices with predefined colours that were allocated first, did not change, and the cardinality of the colour classes are approximately equal[1].

A two-phase heuristic algorithm is used to obtain an initial feasible solution. The first phase in which a proper colouring only is sought, is discussed in §3.2.1, whereafter the second phase of obtaining an equitable colouring is discussed in §3.2.2.

3.2.1 First phase

Given a timetabling graph G, the first phase of the algorithm consists of finding a proper coloured graph G_c of the partially coloured graph G_p. The partially coloured graph G_p is obtained from G by allocating the vertices that need predefined colours, these respective colours. After this is done, G_p is inspected to ensure that the partial colouring is feasible. It could, for instance, be infeasible due to two or more adjacent vertices having the same colour within G_p. This will happen when two or more examination papers must be written in the same timeslot, where in fact there exists at least one student within the dataset that must write two or more of these examination papers. If the partial colouring proves to be infeasible, a message is displayed indicating which examination papers are causing this infeasibility. The pseudocode for this part of obtaining an initial solution, is given in Algorithm 1.

Algorithm 1: `Initial solution, Phase 1`

Input:
G_p, a partially coloured graph of G.
Set $\mathcal{T}$ containing all available colours.
Set $\mathcal{P}$ containing the vertices that have already been coloured.
Set $\mathcal{M}$ containing the vertices that must still be coloured.
Output: A proper coloured graph G_c of the partially coloured graph G_p.

while $|\mathcal{M}| \neq 0$ **do**
  $\mathcal{D} \leftarrow \emptyset$
  for *all vertices in* $\mathcal{M}$ **do**
    $u \leftarrow$ Current vertex
    $f(u) = |\mathcal{T}| - \varrho(u)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{f(u)\}$
  end
  $v \leftarrow$ the vertex with the smallest value in $\mathcal{D}$
  if $f(v) = 0$ **then**
    terminate algorithm - no solution could be found
  end if
  Assign the element with the smallest index in $\mathcal{T} \setminus \mathcal{N}_v$ to vertex v, where $\mathcal{N}_v$ is the set containing all the colours of vertex v's neighbours.
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{v\}$
  $\mathcal{M} \leftarrow \mathcal{M} \setminus \{v\}$
end while
$G_c \leftarrow G_p$

Algorithm 1 takes the partially coloured graph G_p as input, together with the set $\mathcal{T}$ containing all available colours, the set $\mathcal{P}$ of vertices that have already been coloured in G_p and the set $\mathcal{M}$ of vertices that still need to be coloured. Note that $|\mathcal{T}|$ is equivalent to the total number of timeslots within the examination period.

[1]Note that Requirement 3 in §3.1, *i.e.* the constraint of some vertices having the same colour, is ignored, since the vertices representing those examination papers are merged, as described in §2.3, before an initial solution is obtained.

The first step of Algorithm 1 is to determine a colouring difficulty for all the vertices in $\mathcal{M}$, which is done in Step 2. The colouring difficulty $f(u)$ of a vertex u is determined by $f(u) = |\mathcal{T}| - \varrho(u)$, where $\varrho(u)$ is the saturation degree of vertex u. Thus, the colouring difficulty $f(u)$ of vertex u is the number of colours that is still available to colour vertex u at this stage of the heuristic without causing infeasibility. Since the more colours are available, the easier it might be to colour a particular vertex, the vertex v with the *lowest* colouring difficulty is chosen to be coloured first. This is done in Step 3 of Algorithm 1, where ties in colouring difficulty result in choosing the vertex with the lowest index. If $f(v) = 0$ in Step 4, it means that vertex v must still be coloured, but no colour is available to colour vertex v without causing infeasibility. The algorithm will then terminate in Step 5, as no feasible proper colouring of G_p could be found.

If $f(v) \neq 0$, the colouring in Step 7 is executed, where vertex v is assigned the colour with the smallest index number in the set $\mathcal{T} \setminus \mathcal{N}_v$, where $\mathcal{N}_v$ is the set containing all the colours of vertex v's neighbours. Note that this algorithm cannot produce an infeasible solution, since a vertex v will only be coloured by colours not used by its neighbours. After the vertex v has been given a colour in Step 7, vertex v is added to the set $\mathcal{P}$ and removed from the set M in Steps 8 and 9. This iterative process is terminated when $\mathcal{M} = \emptyset$, or when any of the vertices in $\mathcal{M}$ could not be coloured. The algorithm assigned colours to vertices in G_p, G_p is a proper coloured graph if $\mathcal{M} = \emptyset$, and is renamed to G_c in Step 11. An example of how Algorithm 1 generates a proper coloured graph G_c of a partially coloured graph G_p is explained in Example 3.2.1.

Example 3.2.1. Suppose a proper coloured graph G_c of the partially coloured graph G_p in Figure 3.5 is sought. Furthermore, let the four colours in Table 3.4 be the only colours available to colour the vertices in G_p.

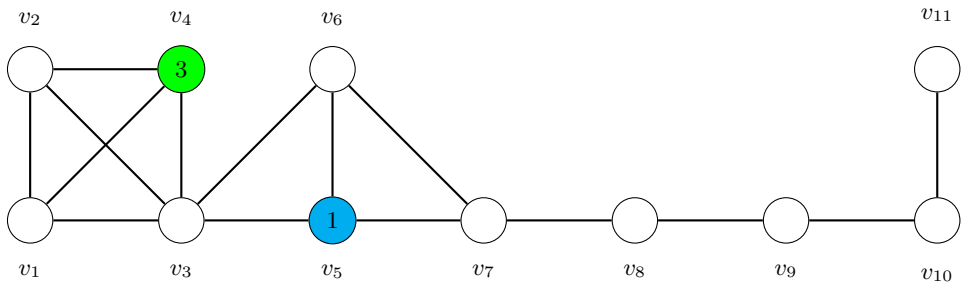


FIGURE 3.5: *The partially coloured graph G_p used in Example 3.2.1.*

Index	Colour
1	Blue
2	Red
3	Green
4	Yellow

TABLE 3.4: *The indices of the colours available for use in Example 3.2.1.*

From Figure 3.5, $\mathcal{T} = \{1, 2, 3, 4\}$, $\mathcal{P} = \{v_4, v_5\}$ and $\mathcal{M} = \{v_1, v_2, v_3, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$, and these sets are the input for Algorithm 1. Each iteration of Algorithm 1 starts with calculating the colouring difficulty $f(u) = |\mathcal{T}| - \varrho(u)$ for every vertex $u \in \mathcal{M}$, where $|\mathcal{T}| = 4$ for this example. During the first iteration $f(v_1) = 4 - 1 = 3$, $f(v_2) = 4 - 1 = 3$, $f(v_3) = 4 - 2 = 2$, $f(v_6) = 4 - 1 = 3$, $f(v_7) = 4 - 1 = 3$ and $f(v_8) = f(v_9) = f(v_{10}) = f(v_{11}) = 4 - 0 = 4$. Vertex v_3 is chosen to be coloured first, since the colouring difficulty $f(v_3)$ is the smallest value

amongst all the colouring difficulties. The vertex v_3 will be coloured with the colour with the smallest index in $\mathcal{T} \setminus \mathcal{N}_{v_3}$, where $\mathcal{N}_{v_3}$ contains the colours of vertex v_3's neighbours. Index 2 (corresponding to red) is the smallest index in the set $\{1,2,3,4\} \setminus \{1,3\} = \{2,4\}$. Thus, v_3 will be coloured red. The resulting graph can be seen in Figure 3.6. Vertex v_3 is then added to set $\mathcal{P}$ and removed from set $\mathcal{M}$.

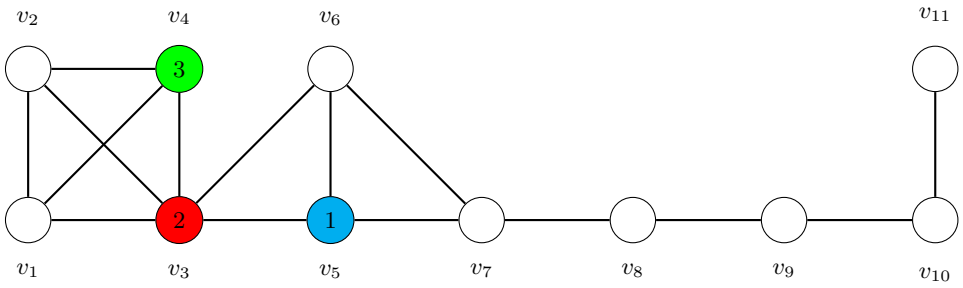


FIGURE 3.6: *The partially coloured graph G_p used in Example 3.2.1 after the first iteration.*

At the next iteration, $f(v_1) = f(v_2) = f(v_6) = 2$ is the smallest colouring difficulty amongst all colouring difficulties. Since more than one vertex has the smallest colouring difficulty, the vertex with the smallest index is chosen to be coloured first. Thus vertex v_1 is chosen to be coloured with blue, since 1 is the smallest index in $\{1,2,3,4\} \setminus \{2,3\} = \{1,4\}$. After vertex v_1 is coloured, $f(v_2) = 1$ will be the smallest colouring difficulty and v_2 will be coloured with yellow, since it is the only colour available to colour v_2. Next, vertex v_6 has the smallest colouring difficulty and will be coloured green, since 3 is the smallest index in the set $\{1,2,3,4\} \setminus \{1,2\} = \{3,4\}$. Vertex v_7 will then be coloured red, followed by vertex v_8 being coloured with blue, vertex v_9 coloured red again, vertex v_{10} being coloured with blue and finally vertex v_{11} being assigned the colour red. The resulting proper coloured graph can be seen in Figure 3.7. ■

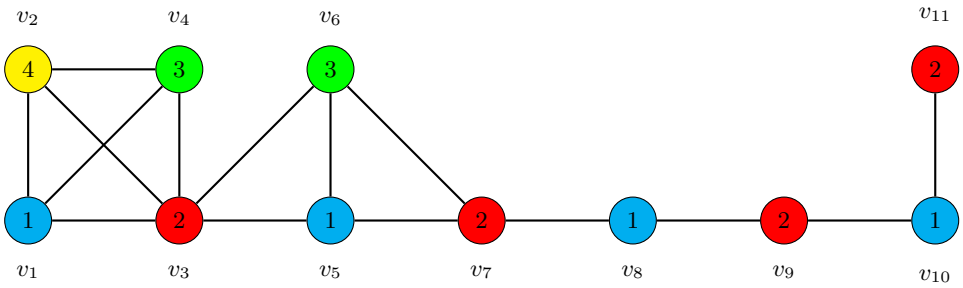


FIGURE 3.7: *The proper coloured graph G_c obtained in Example 3.2.1.*

3.2.2 Second phase

The second phase of the algorithm consists of trying to find an equitable coloured graph G_e of graph G_c obtained from the first phase. The pseodocode for this part of obtaining an initial solution can be seen in Algorithm 2.

The proper coloured graph G_c of graph G is the input for Algorithm 2, together with the total number of examination papers, N, that must be scheduled, the set $\mathcal{T}$ containing all available colours, the set $\mathcal{W}$ containing all the vertices with fixed colours and the maximum number of iterations I that may be used. Note that $p(G_c)$ is not necessarily the same as N, because vertices

Algorithm 2: `Initial solution, Phase 2`

Input:
G_c, a proper coloured graph of G.
N, the total number of examination papers that must be scheduled.
Set $\mathcal{T}$ containing all available colours.
Set $\mathcal{W}$ containing all vertices with predefined, fix colours.
I, the maximum number of iterations used.
Output: An approximation to an equitable coloured graph G_e of graph G_c.

1: $\epsilon = \frac{N}{|\mathcal{T}|}$
2: $\mathcal{S} \leftarrow \emptyset$
3: **for** $m = 1, ..., |\mathcal{T}|$ **do**
4: $\quad \mathcal{C}_m \leftarrow$ All vertices in G_c with colour m
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{C}_m\}$
6: **end for**
7: *iterations* $\leftarrow 1$
8: **while** *iterations* $\neq I$ **do**
9: $\quad$ **if** $|\mathcal{C}_k| = \lceil \epsilon \rceil$ or $|\mathcal{C}_k| = \lfloor \epsilon \rfloor$, $\mathcal{C}_k \in \mathcal{S}$ **then**
10: $\quad\quad$ *terminate algorithm* - an equitable colouring is found
11: $\quad$ **end if**
12: $\quad \mathcal{S}_1 \leftarrow \{\mathcal{C}_k \mid |\mathcal{C}_k| \geq \lceil \epsilon \rceil, \mathcal{C}_k \in \mathcal{S}\}$; Sort $\mathcal{S}_1$ in descending order of $|\mathcal{C}_k|$
13: $\quad \mathcal{S}_2 \leftarrow \{\mathcal{C}_k \mid |\mathcal{C}_k| \leq \lfloor \epsilon \rfloor, \mathcal{C}_k \in \mathcal{S}\}$; Sort $\mathcal{S}_2$ in ascending order of $|\mathcal{C}_k|$
14: $\quad$ *stop* $\leftarrow FALSE$
15: $\quad i \leftarrow 0$
16: $\quad$ **while** *stop* $=$ *FALSE* **and** $i < |\mathcal{S}_1|$ **do**
17: $\quad\quad i \leftarrow i + 1$
18: $\quad\quad j \leftarrow 0$
19: $\quad\quad$ **while** *stop* $=$ *FALSE* **and** $j < |\mathcal{S}_2|$ **do**
20: $\quad\quad\quad j \leftarrow j + 1$
21: $\quad\quad\quad k \leftarrow 0$
22: $\quad\quad\quad$ **while** *stop* $=$ *FALSE* **and** $k < |\mathcal{C}_i|$ **do**
23: $\quad\quad\quad\quad k \leftarrow k + 1$
24: $\quad\quad\quad\quad u \leftarrow v_k$ where $v_k \in \mathcal{C}_i$ and $\mathcal{C}_i \in \mathcal{S}_1$
25: $\quad\quad\quad\quad$ **if** $u \notin \mathcal{W}$ **and** $\mathcal{C}_i \leftarrow \mathcal{C}_i \setminus \{u\}$ **and** $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{u\}$ where $\mathcal{C}_j \in \mathcal{S}_2$
26: $\quad\quad\quad\quad$ result in a proper colouring **then**
27: $\quad\quad\quad\quad\quad \mathcal{C}_i \leftarrow \mathcal{C}_i \setminus \{u\}$; $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{u\}$
28: $\quad\quad\quad\quad\quad$ *iterations* $\leftarrow$ *iterations* $+ 1$; $\mathcal{S} \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$; *stop* $\leftarrow TRUE$
29: $\quad\quad\quad\quad$ **end if**
30: $\quad\quad\quad$ **end while**
31: $\quad\quad$ **end while**
32: $\quad$ **end while**
33: **end while**
34: $G_e \leftarrow G_c$

could have been merged, resulting in the number of vertices in G_c being unequal to the number of examination papers.

The algorithm commences in Step 1 by calculating the (fractured) ideal cardinality ϵ of the colour classes in the case of an equitable colouring[2]. This is done by dividing the number of examination papers N by the total number of colours $|\mathcal{T}|$ used. In other words, if one wants the same number of examination papers per timeslot, the number of examination papers should be divided by the total number of available timeslot. Since ϵ is seldom an integer number, in the case of an equitable colouring, the cardinality of the colour classes will be equal to either $\lfloor \epsilon \rfloor$ or $\lceil \epsilon \rceil$.

[2]Note that in this project, the cardinality of a colour class, *i.e.* $|\mathcal{C}_m|$, is the total number of examination papers represented by the vertices coloured with colour m. This means that the cardinality of a colour class is not necessarily equal to the number of vertices in the colour class, due to the merging of vertices before an initial solution was sought.

The next step is to store the colour classes $\mathcal{C}_m$ of all colours $m \in \mathcal{T}$ in a new set of sets $\mathcal{S}$ (Steps 3 to 6). The algorithm then proceeds with an iterative process between Steps 8 and 33, of identifying a vertex in a colour class with a cardinality greater or equal to $\lceil \epsilon \rceil$ and then tries to recolour the vertex with the colour of a colour class with a cardinality less or equal to $\lfloor \epsilon \rfloor$. By doing this, G_c should tend towards being an equitable coloured graph. However, before the process of seeking a vertex to recolour, a check is executed in Step 9 to determine whether an equitable colouring has already been obtained. If an equitable colouring has indeed been obtained, the algorithm terminates in Step 10. Otherwise, after it was determined that an equitable colouring has not yet been obtained, the iterative process of changing a vertex's colour, is started by creating two subsets of $\mathcal{S}$, namely $\mathcal{S}_1$ containing all the elements of $\mathcal{S}$ that have a cardinality greater or equal to $\lceil \epsilon \rceil$, and $\mathcal{S}_2$ containing all the elements of $\mathcal{S}$ with a cardinality less or equal to $\lfloor \epsilon \rfloor$. The set $\mathcal{S}_1$ is sorted in descending order of the cardinalities of the colour classes $\mathcal{C}_k$, while set $\mathcal{S}_2$ is sorted in ascending order of the cardinalities of the colour classes $\mathcal{C}_k$.

During each iteration, the colour class $\mathcal{C}_i$ corresponding to the i'th largest cardinality in $\mathcal{S}_1$, together with colour class $\mathcal{C}_j$ corresponding to the j'th smallest cardinality in $\mathcal{S}_2$, is chosen. After the two colour classes are chosen, the algorithm searches for a vertex $u \in \mathcal{C}_i$ that can be recoloured with the colour corresponding to the colour class $\mathcal{C}_j$ without violating the requirement of a proper colouring, or without recolouring vertex u if u has a predefined, fixed colour. This iterative process of changing vertices' colours will go on until an equitable colouring of G_c is found, or the maximum number of iterations have been reached. After the algorithm is terminated, an approximate equitable coloured graph G_e of G_c is returned. G_e will be the initial solution used in the algorithms to follow in §3.5. An example of obtaining an equitable colouring with Algorithm 2 is illustrated in Example 3.2.2.

Example 3.2.2 (continue from Example 3.2.1)**.** Suppose that an equitable coloured graph G_e of the proper coloured graph G_c in Figure 3.7 is sought, where the colours in Table 3.4 are still the only available colours. Furthermore, from Figure 3.5, vertices v_4 and v_5 have fixed colours, thus $\mathcal{W} = \{v_4, v_5\}$ in Algorithm 2.

The first step of Algorithm 2, is to calculate the ideal cardinality ϵ of the colour classes in the case of an equitable colouring. This means that $\epsilon = \frac{11}{4} = 2.75$, meaning that the cardinality of the colour classes must be either $\lfloor 2.75 \rfloor = 2$ or $\lceil 2.75 \rceil = 3$ in the case of an equitable colouring.

From Figure 3.7 it can be seen that $|\mathcal{C}_1| = 4$, $|\mathcal{C}_2| = 4$, $|\mathcal{C}_3| = 2$ and $|\mathcal{C}_4| = 1$, and thus during the first execution of Step 9 in Algorithm 2 the current colouring failed to be an equitable colouring. Also, subsequently $\mathcal{S}_1 = \{\mathcal{C}_1, \mathcal{C}_2\}$ and $\mathcal{S}_2 = \{\mathcal{C}_3, \mathcal{C}_4\}$. After $\mathcal{S}_1$ and $\mathcal{S}_2$ are created, $\mathcal{S}_1$ is sorted in descending order, while $\mathcal{S}_2$ is sorted in ascending order. Thus, $\mathcal{S}_1$ will remain in the order $\{\mathcal{C}_1, \mathcal{C}_2\}$ and the order of $\mathcal{S}_2$ will be changed to $\{\mathcal{C}_4, \mathcal{C}_3\}$. Next, the stopping criteria is initialised to *FALSE* in Step 14 and the iterative search process to recolour a vertex in the while–loop in Steps 16 to 32 commences. During the first execution of the while–loop between Steps 16 and 32, the blue colour class is selected from $\mathcal{S}_1$, while the yellow colour class is selected from $\mathcal{S}_2$. Then, the first vertex selected from the blue colour class $\mathcal{C}_1 = \{v_1, v_5, v_8, v_{10}\}$ in Step 23 is v_1. From Figure 3.7 it is clear that vertex v_1 can not be recoloured with yellow, because a neighbour vertex v_2 already has the colour yellow assigned to it. Thus, the while–loop between Steps 22 and 30 needs to be repeated. The next two steps of Algorithm 2 applied to graph G_c in Figure 3.7 are given in Table 3.5 and the resulting proper coloured graph is given in Figure 3.8.

During the next iteration of the while–loop between Steps 8 and 33 $|\mathcal{C}_1| = 3$, $|\mathcal{C}_2| = 4$, $|\mathcal{C}_3| = 2$ and $|\mathcal{C}_4| = 2$ and thus an equitable colouring has not yet been reached, because $|C_2| \neq 2$ and $|C_2| \neq 3$. The remainder of the course of the while–loop between Steps 8 and 32 are given in Table 3.6, but note that the indices i and j refer to the sequence in the ordered sets $\mathcal{S}_1$ and $\mathcal{S}_2$ and not the original number representing the colour class. The resulting colouring at the end of

Step	$\mathcal{S}_1$	$\mathcal{S}_2$	i	$\mathcal{C}_i$	colour i	j	$\mathcal{C}_j$	colour j	k	v_k	recolour?
22-30	$\{\mathcal{C}_1, \mathcal{C}_2\}$	$\{\mathcal{C}_4, \mathcal{C}_3\}$	1	$\mathcal{C}_1 = \{v_1, v_5, v_8, v_{10}\}$	blue	1	$\mathcal{C}_4 = \{v_2\}$	yellow	2	v_5	no, $v_5 \in \mathcal{W}$
22-30	"	"	"	"	"	"	"	"	3	v_8	yes

TABLE 3.5: *Steps illustrating the search for a vertex in graph G_c in Figure 3.7 to be recoloured during the first iteration of the while–loop between Steps* 8 *and* 32.

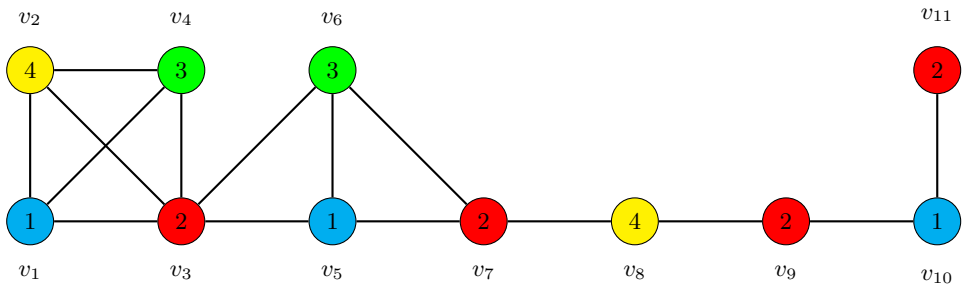


FIGURE 3.8: *An nearly equitable coloured graph obtained in Example 3.2.2 after one iteration of Algorithm 2.*

the steps in Table 3.6 are given in Figure 3.9. For the third iteration of the while–loop between Steps 8 and 33, it holds that $|\mathcal{C}_1| = 3$, $|\mathcal{C}_2| = 3$, $|\mathcal{C}_3| = 3$ and $|\mathcal{C}_4| = 2$ and thus an equitable colouring is obtained and the algorithm terminates at Step 10 with the graph in Figure 3.9 as the final (equitable coloured) graph G_e. ■

Step	$\mathcal{S}_1$	$\mathcal{S}_2$	i	$\mathcal{C}_i$	colour i	j	$\mathcal{C}_j$	colour j	k	v_k	recolour?
12-15	$\{\mathcal{C}_2, \mathcal{C}_1\}$	$\{\mathcal{C}_3, \mathcal{C}_4\}$	0								
16-32	"	"	1	$\mathcal{C}_2 = \{v_3, v_7, v_9, v_{11}\}$	red	1	$\mathcal{C}_3 = \{v_4, v_6\}$	green	1	v_3	no
22-30	"	"	"	"	"	"	"	"	2	v_7	no
22-30	"	"	"	"	"	"	"	"	3	v_9	yes

TABLE 3.6: *Steps of the second iteration of the while loop between Steps* 8 *and* 33*, from Step* 12 *in Example 3.2.1 via Algorithm 2.*

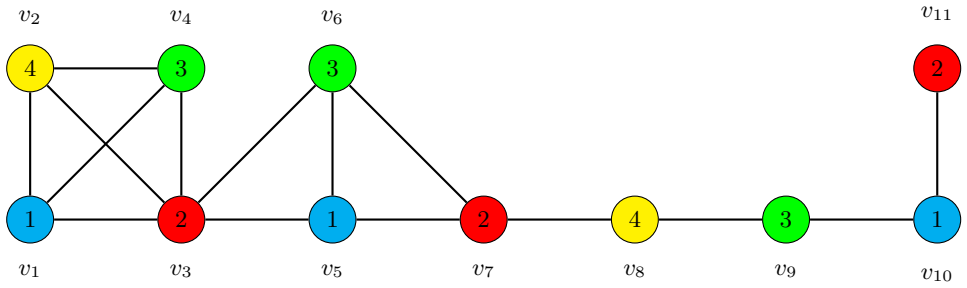


FIGURE 3.9: *The equitable coloured graph G_e obtained in Example 3.2.2.*

3.3 Moves

The moves incorporated in the search algorithms of this project to move from solution to solution in the solution space, are explained in this section. These moves are designed in such a way that only feasible neighbour solutions can be generated, so that the search process always stays within the feasible region of the solution space. Three simple moves are used in the search methods and are explained in §3.3.1, §3.3.2 and §3.3.3, respectively.

3.3.1 Move one module

The first move consists of changing the colour of one vertex within the current colouring. In terms of the examination timetable, this would correspond to moving an examination paper from one timeslot to another timeslot.

The move starts off by selecting a random vertex u in a colour class $\mathcal{C}_m$ of cardinality larger than or equal to $\lfloor \epsilon \rfloor + \alpha(u)$, where $\alpha(u)$ indicates how many vertices were merged to form vertex u. In other words, a vertex u of colour m will only be considered to be recoloured if it does not cause $|\mathcal{C}_m|$ to go below the lower bound of the cardinalities in the case of an equitable colouring.

After a vertex u has been identified, a set $\mathcal{A}_u$ is obtained, where $\mathcal{A}_u$ contains all of the available colours of vertex u that will still result in a proper colouring. A random colour $\ell \in \mathcal{A}_u$ is chosen to be considered as the new colour of vertex u. If $|\mathcal{C}_\ell| + \alpha(u) \leq \lceil \epsilon \rceil$, vertex u will be recoloured with colour ℓ. In other words, if the cardinality of colour class $\mathcal{C}_\ell$ plus the number of vertices represented by vertex u is still lower than the upper bound of the cardinalities in the case of an equitable colouring, vertex u can be recoloured with colour ℓ without causing infeasibility, *i.e.* a solution that is not an equitable colouring.

This move, however, cannot be the only move used in the search algorithms in this project, because complications can arise that will make this move inefficient in exploring the entire solution space. For instance, consider colouring a graph where the cardinality of the largest clique is equal to the number of colours available. After an initial solution is obtained, it will be impossible to change the colours of any of the vertices within the largest clique with this move. This would be because for all vertices u in the clique, the set $\mathcal{A}_u$ will be empty. So even though there are other feasible solutions within the solution space where the vertices within the largest clique have different colours, using only this move will only allow searches through the solution space where the vertices within the largest clique have a fixed colouring. Other moves are thus necessary to compensate for this factor, since effective search algorithms need to be able to explore the entire solution space.

This move provides a way to steer an initial solution that is not an equitable colouring towards being an equitable colouring. This is because the move only changes the colour of a vertex that belongs to a colour class with cardinality larger than the lower bound of an equitable colouring, and changes its colour to a colour corresponding with a colour class of cardinality smaller than the upper bound. This move is illustrated in Example 3.3.1.

Example 3.3.1. Suppose one wants to recolour one of the vertices of the graph G_e of Figure 3.10 in such a way that the resulting colouring is still a feasible solution according to the variant of the ETP of this project. Furthermore, only the four colours in Table 3.4 are available for use, and vertex v_8 has a fixed predefined colour, namely red. In this graph, the label of a vertex indicates the name of the vertex, followed by the number of vertices that were merged to form the vertex (*i.e.* how many examination papers are represented by the vertex).

Since both vertex v_1 and vertex v_{11} represent two examination papers each, and $|p(G_e)| = 12$, a total of 14 examination papers are represented by the vertices in G_e. This means that the cardinality of all colour classes must be either $\lfloor \frac{14}{4} \rfloor = 3$ or $\lceil \frac{14}{4} \rceil = 4$ in the case of a equitable colouring.

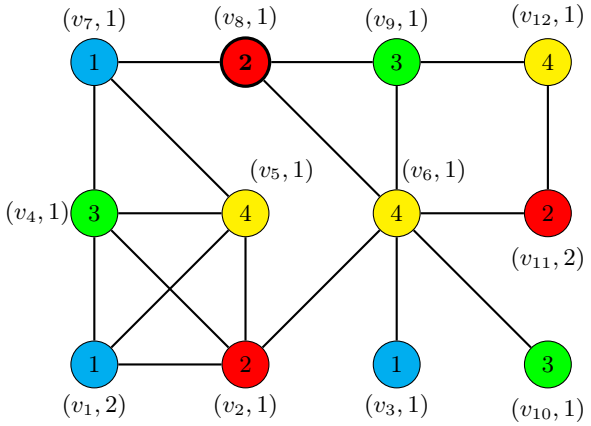


FIGURE 3.10: *The equitable coloured graph G_e used in Example 3.3.1.*

The first step of recolouring a vertex in G_e, is to identify all vertices that may have their colours changed without violating the requirement of an equitable colouring, *i.e.* all vertices v such that the cardinality of the colour class v belongs to will remain 3 or more if v is removed from the colour class. This automatically removes all vertices that are currently in colour classes with cardinality 3 as options to be recoloured. The cardinality of the colour classes (including merged vertices) at the moment is $|\mathcal{C}_1| = 4$, $|\mathcal{C}_2| = 4$, $|\mathcal{C}_3| = 3$ and $|\mathcal{C}_4| = 3$. Thus vertices coloured green or yellow may not be recoloured. Furthermore, both vertex v_1 and vertex v_{11} may not have their colours changed, as this would result in the cardinality of their respective colour classes being 2. Thus, the only remaining options are vertices v_2, v_3, v_7 and vertex v_8, but vertex v_8 has a fixed predefined colour, so this only leaves vertex v_2, v_3 and vertex v_7 as options to be recoloured. One of these three vertices is then chosen at random, say v_7. However, the set $\mathcal{A}_{v_7}$ (the set containing all available colours that may be used to recolour v_7 while a proper colouring is maintained) is an empty set, since the saturation degree $\varrho(v_7) = 3$. This means that vertex v_7 cannot be recoloured without breaking feasibility. The next random vertex from the set $\{v_2, v_3\}$ is then considered, say v_3. For this vertex, $\mathcal{A}_{v_3} = \{2, 3\}$, meaning that vertex v_3 may be recoloured with either red or green while the resulting colouring will remain a proper colouring. However, $|\mathcal{C}_2| = 4$, so vertex v_3 may not be recoloured with red, since this will make the cardinality of the red colour class larger than 4. Thus, 2 is removed from $\mathcal{A}_{v_3}$, leaving 3 as the only possibility. Vertex v_3 will thus be recoloured with the colour green to form a neighbour solution of the graph G_e is Figure 3.10. ■

3.3.2 Swap two colour classes

The second move consists of identifying two random colour classes $\mathcal{C}_m$ and $\mathcal{C}_\ell$, and swapping the vertices in $\mathcal{C}_m$ with the vertices in $\mathcal{C}_\ell$. In terms of examination papers within the timetable, this would translate to swapping all of the examination papers within one timeslot with all of the examination papers within another timeslot.

It is possible to prove, with the use of Kempe chains [1], that this swapping of colours will still

result in a proper colouring. Furthermore, this move does not change the cardinality of the colour classes either. It is, however, important not to change the colour of vertices with a fixed predefined colour. Thus, for this move a colour class that contains vertices with fixed colours will never be chosen as one of the colour class to be swapped.

This move, in combination with the move described in §3.3.1, provide an efficient way of exploring other parts of the solution space than the current solution. The shortage of the move described in §3.3.1 due to the complications when trying to change the colours of vertices within the large cliques, can be overcome with the swapping of two entire colour classes. Thus, during the move described in this section the colours of vertices within cliques may be changed, as long as those vertices' colours are not predefined to be fixed. An example of using this move is given in Example 3.3.2.

Example 3.3.2. Suppose one wants to swap all the vertices in two randomly selected colour classes of graph G_e in Figure 3.10 (repeated in Figure 3.11(a)) to obtain a feasible neighbour solution of G_e. Furthermore, let vertex v_8 have a fixed predefined colour, namely red.

This move starts off by selecting two random colour classes without any vertices with fixed colours. Thus, in this example, the red colour class will not be selected. From the remaining blue, yellow and green colour classes, let blue and yellow be chosen arbitrarily. The resulting neighbour solution of G_e after the blue and yellow colour classes were swapped, can be seen in Figure 3.11(b). This solution is still a feasible solution of the ETP variant used in this project. Furthermore, note that this move managed to swap the colours of vertices v_1 and v_5 within the clique of size 4 formed by vertices v_1, v_2, v_4 and v_5, with only 4 colours available to use. This would have been impossible with the move described in §3.3.1, since $\mathcal{A}_{v_1} = \mathcal{A}_{v_2} = \mathcal{A}_{v_4} = \mathcal{A}_{v_5} = \emptyset$, so none of these vertices could have been recoloured without breaking feasibility. ■

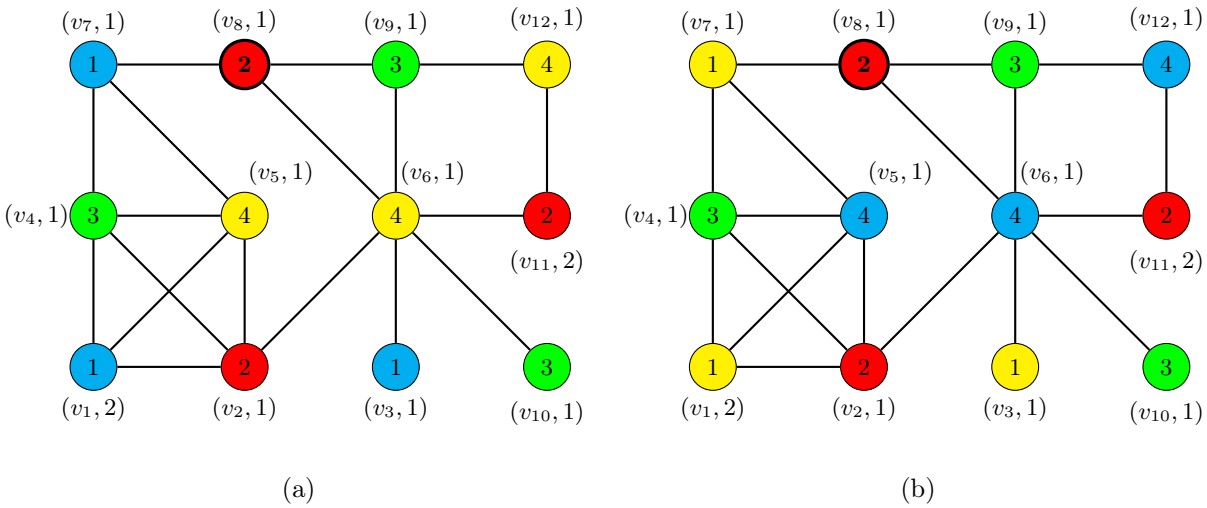


FIGURE 3.11: *(b) The resulting neighbour solution of graph G_e in (a) after the yellow and blue colour classes have been swapped in Example 3.3.2.*

The swapping of two colour classes can significantly change the cost function value when moving from a current solution to a neighbour solution, as swapping all of the examination papers within two timeslot affect a lot of students' examination timetables. This can be advantageous at the early stages of the search algorithms, as large improvements could potentially be made on the current solution in terms of the cost function value. On the other hand, this move might be

ineffective in the later stages of the search algorithms, because at later iterations the current solution could be near optimal, causing large changes in the solution to, most likely, worsen the solution in terms of the cost function value. Thus, the new solution, obtained via this move will be rejected at later iterations of the search algorithms, since the acceptance levels of accepting a new, worse solution, are restricted during later iterations of the search algorithms. Thus, using this move too often during later iterations of the search algorithms could increase execution time unnecessarily.

3.3.3 Swap two modules

The third and final move consists of swapping the colours of two distinctly coloured vertices. In terms of the examination papers within the timetable, this translates to swapping the timeslots of two examination papers that are in different timeslots.

The selection of the two vertices to swap colours are determined randomly. This is done by creating two identical sets $\mathcal{V}_1$ and $\mathcal{V}_2$ that contains all the vertices of the graph. The vertices with fixed predefined colours are then removed from these lists, since their colours may never be changed. Both lists are then ordered in two separate random orders. The first vertex, say u_1, in the list $\mathcal{V}_1$ as well as the first vertex, say v_1, in the list $\mathcal{V}_2$ are chosen. If vertex u_1 has the same colour as vertex v_1, the next vertex u_2 in list $\mathcal{V}_1$ will be chosen and again the two vertices are checked whether they have different colours[3]. This process will be repeated until two vertices with different colours are chosen. Next, a test is executed to determine if the colours of these two vertices can be swapped to form a new solution that is still a feasible solution. If the colours of these two vertices cannot be swapped, the next vertex in the list $\mathcal{V}_1$ together with vertex v_1 are considered. If all of the vertices in list $\mathcal{V}_1$ have been considered in combination with v_1 and no new feasible solution could be found, this process will repeat in a iterative process using the next vertex, v_2, in the list $\mathcal{V}_2$. As soon as a new feasible solution is found, the iterative process stops and the new solution is used in the next step of the search algorithm.

This move, like the move described in §3.3.2, can overcome the shortage of the move described in §3.3.1 in terms of changing the colours of vertices within some of the large cliques in the graph. Since the current solution will always be a proper colouring, all the vertices within a clique will have different colours. Thus, any two vertices within a clique could swap colours without causing infeasibility in the clique itself. So, when swapping the colours of two vertices within a clique of a graph, it is only necessary to check whether the two vertices have other neighbouring vertices outside the clique that have the same colour as the colour these two will be recoloured with. This move thus makes it possible to change the colours of vertices within large cliques, as long as their colours are not fixed.

The move of swapping two modules as described in this section, should cause a smaller change in the cost function value when the search algorithm moves from the current solution to a neighbour solution. This could be useful at later stages of a search algorithm to fine-tune the solution, unlike the move described in §3.3.2.

Example 3.3.3. Suppose one wants to swap the colours of two distinct vertices of the graph in Figure 3.11(b), as to generate a feasible neighbour solution. Again, let vertex v_8 have a fixed predefined colour, namely red.

The first step is to create two identical lists $\mathcal{V}_1$ and $\mathcal{V}_2$ that contain all the vertices of the graph,

[3]Note that one does not also have to check whether $u_1 \in \mathcal{V}1$ and $v_1 \in \mathcal{V}_2$ are the same vertex, since the vertex has the same colour in both lists.

followed by removing vertices with fixed colours, thus

$$\mathcal{V}_1 = \mathcal{V}_2 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_9, v_{10}, v_{11}, v_{12}\}.$$

Next, the two lists are ordered in two separate random orders, say

$\mathcal{V}_1 = \{v_9, v_1, v_4, v_5, v_7, v_{11}, v_3, v_6, v_{10}, v_2, v_{12}\}$ and $\mathcal{V}_2 = \{v_2, v_{12}, v_9, v_5, v_4, v_7, v_3, v_1, v_{10}, v_{11}, v_6\}$.

The first vertex in list $\mathcal{V}2$, *i.e.* v_2, is considered to have its colour swapped with the first vertex in list $\mathcal{V}1$, *i.e* v_9. It can be seen from Figure 3.11(b) that this swapping of colours will not result in a feasible solution, as adjacent vertices v_8 and v_9 will both be coloured red. The next vertex in $\mathcal{V}1$ is v_1. If vertices v_1 and v_2 have their colours swapped, it will result in a proper colouring, but this will not be a feasible colouring since the cardinality of the red colour class will be 5 after the colour swap. Thus, the next vertex in $\mathcal{V}1$, v_4, is considered together with vertex v_2. This swap will result in a feasible neighbour solution, since a proper colouring is maintained while the cardinality of the colour classes remain within bounds. Note that this move, just like the move described in §3.3.2, managed to swap the colours of two vertices within a relatively large clique. ■

3.4 Cost function

Search algorithms need an objective function to be able to evaluate solutions within the search space. In this project, a cost function that reflects how the students' examination papers are spaced out within a given examination timetable, is used. In this regard, the number of papers written shortly after one another for each student, is minimised.

To the best knowledge of the author, no efficient cost function for spacing out students' examinations for this variant of the ETP is available in literature. The best and most relevant cost function that could be found in literature is explained and discussed in §3.4.1, followed in §3.4.2 by an explanation of the cost function that is derived for this project.

3.4.1 Cost function of the 2nd International Timetabling Competition

The cost function used in the 2nd International Timetabling Competition [30] is still used by many researchers to space out the students' examinations in the variant of the ETP studied by the specific researchers [3]. This cost function, however, has some limitations. In this cost function a value must be chosen for a parameter g which defines how close a student's examination papers need to be scheduled before a penalty is assigned in the cost function. For instance, if $g = 3$ it would mean that if any two examination papers have to be written by a student within the space of three consecutive timeslots, the total penalty of the cost function will increase by 1.

However, this cost function *can* be inefficient. To illustrate, suppose there is an examination period containing 10 timeslots and let $g = 3$. First, let student A have four distinct examination papers to write that are scheduled as seen in the second row of Table 3.7.

The total penalty for student A will be 0 since no pair of examination papers are scheduled to be written within the space of three consecutive timeslots. If any of the student's examination papers were to shift to another timeslot, a penalty of at least 1 will definitely be added to the cost function. Thus, the cost function of the 2nd International Timetabling Competition works perfectly for this example, because a perfect spread of examination papers over the entire examination period is obtained when the cost function is minimised. Now, let student B be a

Timeslot	1	2	3	4	5	6	7	8	9	10
Student A's examination papers	1			2			3			4
Student B's examination papers		1			2			3		

TABLE 3.7: *Example of two students' examination timetables where there are* 10 *timeslots in the examination period.*

student who is writing three distinct examination papers during the same examination period, as scheduled in the third row of Table 3.7.

Again, the examination timetable for student B in Table 3.7 also produces a penalty of 0 since none of student B's examination papers are written within the space of three consecutive timeslots. However, the spread of the examination papers for student B can be improved by moving the third examination paper to Timeslot 10 and the first examination paper to Timeslot 1. It is thus concluded that the cost function used in the 2nd International Timetabling Competition lacks the ability to adapt to how many examination papers are written for distinct students. This phenomenon could cause the cost function to be bias towards students with a certain number of examination papers. In this project, however, it is aimed to provide fairness towards all students, regardless of the number of examination papers that should be written by each student. It is for this reason that a new cost function is sought.

3.4.2 Deriving a cost function

In order to derive a cost function to evaluate how evenly each student's examination papers are scheduled through the examination period, an adaptation to the colouring needs to be made. Until now, each colour is associated with an actual timeslot in which examination papers will be written and the number of timeslots (colours) is predefined. At Stellenbosch University, there are two examination timeslots per day, namely one morning and one afternoon slot, from Monday to Saturday. Thus, there are a total of 12 colours used for every week. However, when spacing a student's examination papers through the entire examination period, Sundays should also be taken into consideration, since Sundays provide additional study time. Hence, a simple function was developed to appropriately update the colours of the graph to include Sundays. These updates are illustrated in Table 3.8, where row 3 represents the original corresponding colours for the examination timeslots and row 4 represents the updated colours. As soon as the cost function has been evaluated, a simple function converts the colours back to their normal form of excluding Sundays in order for the algorithm to proceed with another move. These updated colours will thus only be taken into account when the cost function is being evaluated, and not at all during the moves described in §3.3. For the rest of this section the updated colours that include Sundays is used when referring to colours or timeslots.

Day	Tuesday		Wednesday		Thursday		Friday		Saturday		Sunday		Monday	
Time	09:00	14:00	09:00	14:00	09:00	14:00	09:00	14:00	09:00	14:00	09:00	14:00	09:00	14:00
Colour	0	1	2	3	4	5	6	7	8	9			10	11
Updated Colour	0	1	2	3	4	5	6	7	8	9	10	11	12	13

TABLE 3.8: *Illustration of updating the colours corresponding to the timeslots, as to include the number of Sundays within the examination period for the purpose of evaluating the cost function.*

As discussed in §3.4.1, in an efficient cost function the number of examination papers that distinct students have to write, have to be considered. Let r_i denote the total number of distinct

examination papers that a unique student i must write and let Υ denote the total number of timeslots available in the examination period, including Sundays (*i.e.* $\Upsilon = |\mathcal{T}| + 2x$, where x is the number of Sundays in the examination period and $|\mathcal{T}|$ is the total number of colours available). There are a total of $\Upsilon - r_i$ timeslots within the examination period that student i is not writing any examination papers. If λ_i denote the perfect spacing between student i's examination papers, then $\lambda_i = \frac{\Upsilon - r_i}{r_i - 1}$, where the denominator is the number of timeslots between student i's examination papers. For example, let a student C have a total number of 4 distinct examination papers that must be scheduled and let the examination period consist of 13 timeslots. Thus, there will be $13 - 4 = 9$ timeslots in the examination period that student C is not writing any examination papers and there will be $4 - 1 = 3$ timeslots between each pair of his/her examination papers. Hence, the perfect spacing between student C's examination papers would be $\lambda_C = \frac{\Upsilon - r_C}{r_C - 1} = \frac{13-4}{4-1} = 3$ timeslots as illustrated in Table 3.9.

Timeslot	1	2	3	4	5	6	7	8	9	10	11	12	13
Examination paper	1				2				3				4

TABLE 3.9: *Example of the perfect spacing of student C's examination papers over the course of the entire length of the examination period.*

However, λ_i will not always be an integer in which case the timeslots between scheduled examination papers of a perfect spacing will be either $\lfloor \lambda_i \rfloor$ or $\lceil \lambda_i \rceil$, with at least one spacing being $\lceil \lambda_i \rceil$. To illustrate, let student D have a total number of 4 examination papers that must be scheduled over the course of 11 examination timeslots. In this case $\lambda_D = \frac{\Upsilon - r_D}{r_D - 1} = \frac{11-4}{4-1} = 2.33$. This means that the timeslots between student D's examination papers are either $\lfloor \lambda_D \rfloor = 2$ or $\lceil \lambda_D \rceil = 3$ with at least one spacing between papers being $\lceil \lambda_D \rceil = 3$. Three examples of student D's examination timetable can be seen in Table 3.10. The top and middle timetable within Table 3.10 are optimally spaced out, but the bottom timetable is not optimal because it does not contain at least 1 set of timeslots between examination papers of size $\lceil \lambda_D \rceil = 3$.

Timeslot	1	2	3	4	5	6	7	8	9	10	11
Schedule 1	1				2			3			4
Schedule 2	1			2				3			4
Schedule 3	1			2			3			4	

TABLE 3.10: *Three examples of scheduling student D's examination papers.*

Next, the penalties of a suboptimal spacing needs to be determined. The cost function $C(G_e)$ of the current solution G_e consists of the sum over all students' penalties, thus

$$C(G_e) = \sum_{i=1}^{t} T(i), \tag{3.1}$$

where $T(i)$ denotes the total penalty for student i, and there is a total number of t students. The function $T(i)$ will assign penalties according to the size of the timeslots between student i's consecutive examination papers. As before, Υ is the total number of timeslots available in the timetable, while r_i is the total number of examination papers student i have to write. Let $n_{i,j}$ denote the timeslot in which student i's jth examination paper is scheduled, then the number of timeslots between two consecutive examination papers for student i is $n_{i,j+1} - n_{i,j} - 1$. If

this spacing differs from the perfect spacing $\lambda_i = \frac{\Upsilon - r_i}{r_i - 1}$, it should be penalised accordingly. The difference between the perfect spacing λ_i and the actual number of timeslots between papers j and $j+1$ can be computed by $|n_{i,j+1} - n_{i,j} - 1 - \lambda_i|$. If $n_{i,j+1} - n_{i,j} - 1 - \lambda_i \geq 0$ the number of timeslots between student i's j'th and $(j+1)$th examination papers is either the perfect size or larger than the perfect size. On the other hand, if $n_{i,j+1} - n_{i,j} - 1 < \lambda_i$, the number of timeslots between student i's jth and $(j+1)$th examination papers is smaller that the perfect spacing and must be penalised in the function $T(i)$. Furthermore, the larger the difference between $n_{i,j+1} - n_{i,j} - 1$ and λ_i, the larger the penalty must be. The intensity of the penalty is done via the input parameter μ in

$$T(i) = \sum_{j=1}^{r_i - 1} \begin{cases} |n_{i,j+1} - n_{i,j} - 1 - \lambda_i|^{\mu} & \text{if } (n_{i,j+1} - n_{i,j} - 1) < \lambda_i, \quad i = 1, 2, ..., t \\ 0 & \text{if } (n_{i,j+1} - n_{i,j} - 1) \geq \lambda_i, \quad i = 1, 2, ..., t, \end{cases} \tag{3.2}$$

where $\mu \geq 1$. It can be observed that the larger the value of μ, the bigger the penalty would be if the number of timeslots between consecutive papers are too small. For example, the size of the spacing between student C's first and second examination paper in the timetable in Table 3.9 is $n_{A,2} - n_{A,1} - 1 = 5 - 1 - 1 = 3$, while the difference between $n_{C,2} - n_{C,1} - 1$ and λ_C is 0, so no penalty is necessary.

For another example to illustrate function (3.2), consider the timetables for students E and F in Table 3.11 and arbitrarily choose $\mu = 2$. It is clear that the number of examination papers $r_E = 4$, the length of the examination period $\Upsilon = 13$ and the prefect spacing $\lambda_E = \frac{\Upsilon - r_E}{r_E - 1} = 3$. The total penalty for student E using function (3.2) is

$$\begin{aligned} T(E) &= \sum_{j=1}^{3} \begin{cases} |n_{E,j+1} - n_{E,j} - 1 - 3|^{\mu} & \text{if } (n_{E,j+1} - n_{E,j} - 1) < 3, \\ 0 & \text{if } (n_{E,j+1} - n_{E,j} - 1) \geq 3, \end{cases} \\ &= |-3|^2 + 0 + |-1|^2 = 10. \end{aligned}$$

Thus, since the size of the spacing between papers 1 and 2 is 3 smaller than the perfect spacing and the spacing between papers 3 and 4 is 1 smaller than the perfect spacing, these spacings are penalised in the function $T(E)$, while the spacing between papers 2 and 3 will not be penalised since it is larger than the perfect spacing.

Timeslot	1	2	3	4	5	6	7	8	9	10	11	12	13
Student E's examination papers	1	2								3			4
Student F's examination papers		1		2			3				4		

TABLE 3.11: *Examination timetables of two students to illustrate the penalty function (3.2).*

If student E's second examination paper is moved to the fourth timeslot, this students' timetable will be more spaced out. In this case one should expect the function $T(E)$ to have a smaller penalty than before, and indeed $T(E) = |-1|^2 + 0 + |-1|^2 = 2$. It would therefore seem as if function (3.2) together with equation (3.1) will suffice as a cost function for our search algorithms. However, this is not quite the case as evident from studying the timetable for student F in Table 3.11.

With $\mu = 2$, the penalty for student F is $T(F) = |-2|^2 + |-1|^2 + 0 = 5$. When student F's fourth examination paper is moved to Timeslot 13, his/her papers are spread over a larger

part of the examination period. Thus, one would expect the new $T(F)$ value to reflect this improvement. When calculating the new penalty, though, the same penalty as the previous timetable is obtained, since $T(F) = |-2|^2 + |-1|^2 + 0 = 5$. The reason for this is because only the spacings that are smaller than the perfect spacing are penalised in the cost function. To incorporate a small reward (negative penalty) for spacings larger than the perfect spacing to the total penalty of student i, a new input parameter δ, where $0 \leq \delta \leq 1$, is introduces in function (3.2) to obtain

$$T(i) = \sum_{j=1}^{r_i-1} \begin{cases} |n_{i,j+1} - n_{i,j} - 1 - \lambda_i|^\mu & \text{if } (n_{i,j+1} - n_{i,j} - 1) < \lambda_i, \quad i = 1, 2, ..., t \\ -\delta(n_{i,j+1} - n_{i,j} - 1 - \lambda_i) & \text{if } (n_{i,j+1} - n_{i,j} - 1) > \lambda_i, \quad i = 1, 2, ..., t \\ 0 & \text{if } (n_{i,j+1} - n_{i,j} - 1) = \lambda_i, \quad i = 1, 2, ..., t. \end{cases} \tag{3.3}$$

Values for parameters μ and δ must be chosen appropriately to prevent the algorithm from preferring an examination timetable in which a student's examination papers are clumped up with one big spacing between one pair of consecutive examination papers. This could happen if δ is too large.

The use of function (3.3) is demonstrated via the examination timetables of students Y and Z. First, consider student Y's examination timetable given in the second row of Table 3.12. Using function (3.3) with $\mu = 2$ and $\delta = 0.5$, the total penalty for student Y is $T(Y) = |-2|^2 + 0 + (-0.5(1)) = 3.5$, since $\lambda_Y = 3$. If student Y's fourth examination paper is moved to Timeslot 13, a smaller penalty of $T(Y) = |-2|^2 + 0 + (-0.5(2)) = 3$ is obtained as expected. Similarly, by moving student Y's third examination paper to the sixth timeslot, the spacing between the third and fourth papers becomes even bigger than before and the spacing between the second and third papers becomes smaller than the prefect spacing. It is expected that this new timetable should have a larger penalty than the original timetable in Table 3.12, which is the case as reflected in the new penalty $T(Y) = |-2|^2 + |-1|^2 + (-0.5(2)) = 4$. However, if δ was chosen to be too large, the function (3.3) may fail. For example, if δ was chosen to be 1, the new penalty would have been $T(Y) = |-2|^2 + |-1|^2 + (-1(2)) = 3$, which would have suggested an improvement on the spread of the timetable, which is not the case.

Timeslot	1	2	3	4	5	6	7	8	9	10	11	12	13
Student Y's examination paper	1		2				3					4	
Student Z's examination paper	1	2	3					4					

TABLE 3.12: *The examination timetables of students Y and Z to illustrate the updated function (3.3).*

Student Z's examination timetable in Table 3.12 is considered next, and $\lambda_Z = 3$. Using function (3.3) with $\mu = 2$ and $\delta = 0.5$, the penalty for student Z is $T(Z) = |-3|^2 + |-3|^2 + (-0.5(1)) = 17.5$. To improve the spread of student Z's timetable, move the third examination paper to the seventh timeslot. It is expected that this new timetable will be better than before, since student Z is not required to write 3 consecutive examination papers like before. However, the new penalty is $T(Z) = |-3|^2 + (-0.5(1)) + |-3|^2 = 17.5$, which suggests the same spread of examination papers as before. Thus, the number of examination papers that a student i is required to write over consecutive timeslots has to be included in the cost function. This leads to

$$C(G_e) = \sum_{i=1}^{t} \big(T(i) + K(i)\big), \tag{3.4}$$

where $K(i)$ is used to penalise *clusters* of consecutive examination papers. Let $c_{i,k}$ be the size of the kth cluster of examination papers for student i. For example, in student Z's timetable in Table 3.12, the size of the first cluster $c_{Z,1} = 3$ and student Z has one cluster of examination papers only. The function for penalising clusters is

$$K(i) = \sum_{k=1}^{\psi(i)} (c_{i,k})^{\rho}, \tag{3.5}$$

where $\psi(i)$ is the total number of cluster in the timetable of student i and $\rho \geq 2$ is an input parameter. If $\rho = 2$, and $\mu = 2$ and $\delta = 0.5$ like before, then for student Z's timetable in Table 3.12, $K(Z) = (c_{Z,1})^2 = 9$ and the total penalty for student Z is $T(Z) + K(Z) = 17.5 + 9 = 26.5$. If student Z's third examination paper is moved to the seventh timeslot, like previously described, two clusters with sizes $c_{Z,1} = 2$ and $c_{Z,2} = 2$ are formed and the penalty for the updated timetable is $T(Z) + K(Z) = 17.5 + (2^2 + 2^2) = 25.5$.

Finally, consider the fact that the total number of examination papers per student can differ. If a student P has a total number of 10 examination papers, the potential of having a relatively large cluster of, say size 5, is possible which may lead to a relatively large $K(P)$. On the other hand, if a student Q has only 4 examination papers, the largest cluster that student Q can have is of size 4, and this would be the student's worst case scenario. If student Q is scheduled to write all 4 examination papers in succession and student P is required to write 5 or more of the 10 examination papers in consecutive timeslots, $K(P)$ would be relatively large compared to $K(Q)$, suggesting student P's timetable is worse than student Q's timetable. However, this is not the case, since this was the worst case scenario of student Q's timetable. Thus cost function (3.4) may benefit student P more than student Q based on the fact that student P has more examination papers. Since fairness towards all students is sought, the cost function (3.4) is adapted to

$$C(G_e) = \sum_{i=1}^{t} \frac{T(i)}{T_{i,w}} + \frac{K(i)}{K_{i,w}}, \tag{3.6}$$

where $T_{i,w}$ denotes the worst case scenario of $T(i)$ for student i and $K_{i,w}$ denotes the worst case scenario of $K(i)$ for student i. Note that any student i will have a total penalty of $0 \leq \frac{T(i)}{T_{i,w}} + \frac{K(i)}{K_{i,w}} \leq 2$, where $\frac{T(i)}{T_{i,w}} + \frac{K(i)}{K_{i,w}} = 0$ is the best spread of student i's examination papers and $\frac{T(i)}{T_{i,w}} + \frac{K(i)}{K_{i,w}} = 2$ is the worst spread, which is when all of his/her examination papers are scheduled in consecutive timeslots. Equation (3.6), together with equations (3.3) and (3.5), is the final cost function that will be used in this project.

3.5 Search algorithms

To generate a timetable in which students' examination papers are as spread out as possible, search algorithms are used to effectively move through the solution space. Two different types of search algorithms are being experimented with in this project. The first one is hill climbing that is discussed in §3.5.1, while the great deluge algorithm is discussed in §3.5.2.

3.5.1 Hill climbing

Hill climbing is a greedy, local search algorithm and is given in pseudocode in Algorithm 3.

Algorithm 3: `Hill climbing`

Input:
G_p, a partially coloured graph of graph G.
Set T containing all available colours.
Set $P1$ containing the vertices that have already been coloured as well as the colours assigned to them.
Set $P2$ containing the merged vertices together with the number of vertices merged into each merged vertex.
Set M containing the vertices that still need to be coloured.
I, the maximum number of iterations used in Algorithm 2.
R, the set of rules on how to choose moves at each iteration.
γ, the maximum number of iterations.
$C(G)$, the cost function derived in §3.4.2.
Output: An approximated solution G_f in which students' examination papers are spread out over the entire examination period.

1: $stop \leftarrow 0$
2: Call Algorithms 1 and 2 to generate an initial solution
3: $current\_solution \leftarrow initial\_solution$
4: **while** $stop \neq \gamma$ **do**
5: $stop \leftarrow stop + 1$
6: $move \leftarrow$ Select a move according to R
7: $neighbour\_solution \leftarrow$ Use $move$ on $current\_solution$ to generate a neighbour solution
8: **if** $C(neighbour\_solution) \leq C(current\_solution)$ **then**
9: $current\_solution \leftarrow neighbour\_solution$
10: **end if**
11: **end while**
12: $G_f \leftarrow current\_solution$

The algorithm commences by generating an initial solution in Step 2 by making use of Algorithms 1 and 2, which is then set as the current solution in Step 3. During each iteration of the while–loop between Steps 4 and 11, a neighbour solution from the current solution is generated by making use of one of the three moves defined in §3.3. The input parameter R is a set of rules used to identify which move will be used in every iteration to generate a neighbour solution from the current solution. These rules could be, for example, to randomly choose between the three moves at each iteration. After a neighbour solution has been generated, the algorithm proceeds in Step 8 to compare the cost function value of the neighbour solution to the cost function value of the current solution. If the cost function value of the neighbour solution is lower than the cost function value of the current solution, a better solution has been found and thus the neighbour solution is accepted as the new current solution. If, however, the cost function value of the neighbour solution is more than the cost function value of the current solution, the neighbour solution is rejected and another iteration of the while–loop is executed. After γ iterations of the while–loop, the algorithm terminates with the last current solution as output in Step 12. Parameters R and γ need some calibration to determine appropriate values for the specific implementation of the algorithm.

3.5.2 The great deluge algorithm

Various researchers [10, 11, 23, 31] had success utilising the great deluge algorithm for their variations of the ETP. Therefore, this algorithm will also be considered in this project. The great deluge algorithm is similar to simulated annealing in the way it accepts new solutions as the current solution, however the great deluge algorithm has an advantage over simulated

annealing in the sense that it has less input parameters that need to be calibrated [11]. The great deluge algorithm is given in pseudocode in Algorithm 4. Two new input parameters are specified in Algorithm 4, namely the initial acceptance level θ and the function $D(\theta)$ that is used to lower θ whenever a neighbour solution is accepted as the new current solution.

Algorithm 4: `The great deluge algorithm`

Input:
G_p, a partially coloured graph of graph G.
Set T containing all available colours.
Set $P1$ containing the vertices that have already been coloured as well as the colours assigned to them.
Set $P2$ containing the merged vertices together with the number of vertices merged into each merged vertex.
Set M containing the vertices that still need to be coloured.
I, the maximum number of iterations used in Algorithm 2.
R, the set of rules on how to choose moves at each iteration.
γ, the maximum number of iterations.
$C(G)$, the cost function derived in §3.4.2.
θ, the initial acceptance level.
$D(\theta)$, the function to lowers θ whenever a neighbour solution is accepted.
Output: An approximated solution G_f in which students' examination papers are spread out over the entire examination period.

1: $stop \leftarrow 0$
2: Call Algorithms 1 and 2 to generate an initial solution
3: $current\_solution \leftarrow initial\_solution$
4: $best\_solution \leftarrow current\_solution$
5: **while** $stop \neq \gamma$ **do**
6: $stop \leftarrow stop + 1$
7: $move \leftarrow$ Select a move according to R
8: $neighbour\_solution \leftarrow$ Use $move$ on $current\_solution$ to generate a neighbour solution
9: **if** $C(neighbour\_solution) \leq \theta$ **do**
10: $current\_solution \leftarrow neighbour\_solution$
11: Use $D(\theta)$ to lower the level of θ
12: **if** $C(current\_solution) \leq C(best\_solution)$ **do**
13: $best\_solution \leftarrow current\_solution$
14: **end if**
15: **end if**
16: **end while**
17: $G_f \leftarrow best\_solution$

The algorithm commences by generating an initial solution in Step 2 via Algorithms 1 and 2, and then this initial solution is set as the first current solution in Step 3. The algorithm keeps track of the best solution that has been found thus far, since worse solutions may be accepted in order to explore other parts of the solution space. Thus, at the start of the algorithm the best solution is also the initial current solution, as reflected in Step 4 of Algorithm 4. The algorithm proceeds with an iterative process between Steps 5 and 16 of exploring the solution space for solutions with a cost function value less than θ. First, a neighbour solution of the current solution is generated in Step 8 using a move selected according to R in Step 7. Next, this solution will be accepted as the new current solution if the cost function value of the neighbour solution is less than or equal to θ, as done in Steps 9 and 10. If a new current solution has been found, the level of θ is lowered by using function $D(\theta)$ in Step 11 and a test is done in Step 12 to test whether the new current solution has a better cost function value than the best solution. If the cost function value of the new current solution is less or equal than the cost function value of the best solution, the best solution is updated in Step 13 to be the current solution. After γ iterations, the algorithm terminates and returns the best solution as output in Step 17.

Before using Algorithm 4, the parameters R, γ, θ and $D(\theta)$ need to be calibrated. The parameters R and γ will be calibrated and tested the same way as the hill climbing algorithm, namely

by running the algorithm several times and recording the results.

Dueck [23] gives an idea on choosing θ and $D(\theta)$ that was efficient in his case. The parameter of θ is always chosen to be the cost function value $C(initial\_solution)$. Whenever a new current solution is found, Dueck updated the function by setting

$$D(\theta) = \theta - \frac{\theta - C(neighbour\_solution)}{\beta}, \tag{3.7}$$

where β is a constant parameter. This function is expected to cause large changes in θ at first, followed by smaller changes as the algorithm progresses. In this project, Dueck's choices of θ and $D(\theta)$ will be used together with a variety of values for β.

CHAPTER 4

Case study

Contents

"Do not focus on money, instead focus on a problem that needs to be solved for the world. Money will follow you as a bi-product."

Manoj Arora (2013)

In this chapter the methods described in §3 is applied to the ETP of Stellenbosch University. The datasets provided by Stellenbosch University, together with the data handling done on these datasets to obtain a timetabling graph, is discussed in §4.1. Next, the calibration of the input parameters of the cost function and the search algorithms is explained in §4.2. A discussion of the results of the different algorithms as well as a comparison of the cost function used in this project with cost functions from literature, are the topics of §4.3.

4.1 Data and the timetabling graph of Stellenbosch University

Having access to the necessary data is prerequisite to solving the ETP for any given university. If a university is unable to provide all the necessary data, it could result in a timetable that is not helpful for that specific university. In some cases, if there is a lack of data, assumptions can be made to simplify the variant of that university's ETP. If these assumptions are realistic, a feasible examination timetable could still be generated by using the incomplete datasets.

For this case study, three types of datasets were provided by Stellenbosch University. The contents of these three datasets, as well as the data handling done on them, are discussed in §4.1.1, §4.1.2 and §4.1.3 respectively. Data handling is done by using a combination of Python 3.0 [38] and Oracle SQL Developer 4.1.1 [37]. The section is concluded in §4.1.4 with the timetabling graph for Stellenbosch University that is set up using the datasets discussed in this section.

4.1.1 List of enrolments

The first list obtained was the list of enrolments at Stellenbosch University for every year since the first semester of 2014 until the last semester of 2017. This dataset contains information on the enrolments of all undergraduate students at the university, where each row in the dataset contains the enrolment of *one* student for *one* specific module. The columns of this dataset are as follows:

- *ACADYEAR*. This column contains the academic year of the enrolment. This will either be 2014, 2015, 2016 or 2017.
- *UNIQUEID*. This column refer to each student enrolled for a module. An anonymous unique ID, consisting of two letters and six numbers, was given to each student for the sake of confidentiality. This change from the original student ID to an anonymous ID was done without loss of information necessary for this project.
- *MODULEENG*. The English name of the module that a student was enrolled for is given in this column.
- *MODCODE*. This column contains the unique module code of the module that was enrolled for. The module code is an unique eight digit number.
- *MODULEYEAR*. This column contains either a 1,2,3,4 or 5, which corresponds to the year level (*i.e.* first year level, second year level, *etc.*) of the enrolled module.
- *UNDERPOSTGRADINDENG*. This column shows whether the enrolled module is a post-graduade module or an undergraduate module. However all entries in this dataset are of undergraduate modules.
- *SEMESTERCODE*. This column contains either a 1,2 or 3. A 1 means that the enrolled module is a first semester module, a 2 means that the module is a second semester module and a 3 means that the module is a year module.
- *SEMESTERENG*. This column contains the same information as the *SEMESTERCODE* column, except that the information is given in words and not numbers. The entries would thus be either First Semester Module, Second Semester Module or Year Module.

- *PRGCODE*. The code for the degree program for which the student is enrolled, is given in this column. The program code is an unique six digit number.
- *PROGRAMNAMEENG*. This column contains the English name of the degree program that the enrolled student follows.

However, the only information from this dataset that is needed to set up an examination timetable, are the academic year, the specific student and the module the student is enrolled for. Furthermore, the unique ID's of the students were renamed in such a way that the first student ID is changed to 0 and the next student ID is 1, and so on. This is done in the hope that it could improve the running time of future searches. Thus, a new dataset was created, using the *ACADYEAR*, *UNIQUEID*, *MODCODE*, and *SEMESTERCODE* columns only, and all the other redundant columns were removed. The first 10 rows of the new dataset with the new student ID's can be seen in Table 4.1.

ACADYEAR	UNIQUEID	MODCODE	SEMESTERCODE
2014	0	55743244	2
2014	0	55743142	2
2014	1	11479144	2
2014	2	53848314	1
2014	2	53848344	2
2014	2	53848354	2
2014	2	53848324	1
2014	3	14109178	3
2014	4	18414144	2
2014	4	18414114	1

TABLE 4.1: *First 10 rows of the modified list of enrolments, containing only the data needed to set up the timetabling graph.*

The new dataset was then used to create eight smaller datasets, one for each semester of 2014 to 2017 using the *ACADYEAR* and *SEMESTERCODE* columns of the new dataset. Note that year modules are included in both semesters, as they could have examination opportunities at the end of both semesters.

The layout of the eight smaller datasets were changed to have a list of student enrolments where all the module enrolments of each unique student is given in the same row. Furthermore, the *ACADYEAR* and *SEMESTERCODE* columns in the smaller datasets were deleted, since each of these datasets represent a specific semester of a specific academic year. These new eight datasets are the final lists of enrolments that are used from this stage onwards. An example of these eight new datasets are represented in Table 4.2, where Table 4.2 list some enrolments for the second semester of 2014.

UNIQUEID	List of modules
0	55743244, 55743142
1	11479144
2	53848344, 53848354
3	14109178
4	18414144

TABLE 4.2: *An example of the final list of enrolments for the second semester of* 2014*, obtained from the original list of enrolments in Table 4.1.*

4.1.2 Final examination timetables

Stellenbosch University's head of timetables [25] provided the final examination timetables for the eight semesters of 2014 to 2018. An example of these datasets can be seen in Table 4.3, which contains the first 10 rows of the final examination timetable used during the second semester of 2017. These datasets contain the following columns:

- *FACT*. This column contains the faculty in which the module is offered.
- *NAME*. The name of the examination paper is given in this column. The first part of each entry consists of the English name of the module, followed by the module's three digit short code as well as the paper number of this module, where $P1$ is for the first examination paper of the module and $P2$ is the second paper.
- *CODE*. This column contains the unique eight digit module code.
- *DATE*. The date on which the examination paper is written is given in this column.
- *TIME*. This column contains the time of day on which the examination paper is written.

FACT	NAME	CODE	DATE	TIME
AGRI	Agronomics_342 P1	55565342	02/11/2017	14:00
AGRI	Agronomics_362 P1	55565362	04/11/2017	09:00
AGRI	Agronomics_454 P1	55565454	27/10/2017	14:00
AGRI	AquaCult_344 P1	46213344	01/11/2017	09:00
AGRI	AquaCult_444 P1	46213444	24/10/2017	14:00
AGRI	ConsEcol_244 P1	55638244	02/11/2017	14:00
AGRI	ConsEcol_344 P1	55638344	09/11/2017	14:00
AGRI	Biometry_242 P1	11061242	26/10/2017	09:00
AGRI	Biometry_242 P2	11061242	26/10/2017	14:00
AGRI	Biometry_342 P1	11061342	28/10/2017	09:00

TABLE 4.3: *The first 10 rows of the final examination timetable for the second semester of 2017.*

Not all modules in the list of enrolments described in §4.1.1 necessarily have an examination paper at the end of the semester. Therefore, the final examination timetable of a particular semester was used to identify all the modules that have examination papers at the end of each semester. Using the module codes in a student's enrolled modules, together with the modules in the *CODE* column of the final examination timetable, the list of examination papers that each student must write could be compiled. An example of the list of examination papers per student can be seen in Table 4.4.

UNIQUEID	List of examination papers
0	Agronomics_342 P1, Agronomics_362 P1, Agronomics_454 P1
1	Biometry_242 P1, Biometry_242 P2
2	AquaCult_344 P1, AquaCult_444 P1

TABLE 4.4: *An example of the lists of examination papers that each student must write.*

4.1.3 Information on the examination papers

Two datasets that contain extra information on the examination papers for the years 2016 and 2017 were also provided. Unfortunately, this data recording was only started in 2016 and thus no information could be given on the examination papers prior to 2016. The two datasets contain the following columns:

- *FACULTY*. This column shows which faculty of the university offers the module.
- *DEPT*. The department that offers the module is displayed in this column.
- *NAME*. The English name of the examination paper is shown in this column.
- *OPPORTUNITY*. This column is used to distinguish whether the examination paper is part of the first or second semester's examination timetable. The column contains either 'Jun' for first semester or 'Nov' for second semester.
- *COMPUTER*. If the examination paper of this module must be written in a computer room, this column contains the name of the computer room.
- *EXIT MODULES*. If the module is a final year module for which external moderation needs to take place, it is indicated with an 'x' in this column.
- *FIXED DATES*. If the examination paper of this module must be written on a fixed time and/or date, it will be given in this column.
- *NAME OF SAME TIME*. The names of any modules for which the examination paper must take place during the same timeslot as the current examination paper, are displayed in this column.
- *NAME OF PRECEDING*. This column lists the names of any modules for which the examination papers must be written before this module's examination paper may be scheduled.
- *OTHER REQUIREMENTS*. Requirements which do not fall in any of the other columns can be found in this column. These requirements are mostly just lecturers' preferences of when they preferred that the examination papers of the modules they teach, must be scheduled.

Using the two *OPPORTUNITY* columns, the two datasets were split into four datasets by splitting the data of each year into a first semester and second semester dataset. All redundant columns were then dropped from the datasets. These columns include the *OPPORTUNITY*, *FACULTY* and *DEPT* columns.

The *COMPUTER* column was dropped in all of the datasets, since at Stellenbosch University the scheduling of venues is done *after* the timetables and the scheduling of the examination rooms is not in the scope of this project. Note, however, that by doing this, an assumption is made that during any timeslot there will be enough computer rooms available to accommodate all students that have to write an examination paper in a computer room during a specific timeslot. This assumption should be in order, since there is a relatively small number of modules that require a computer room for examination purposes.

Examination papers of modules that need external moderation must normally be examined by a certain date, causing the examination paper to be scheduled earlier in the examination period to give enough time for the lecturers to examine the papers. Unfortunately not enough

concrete information was given on the specific time implication of each of these modules on the examination timetable. Thus, the *EXIT MODULES* column was dropped from the datasets. A way of dealing with these type of modules in an examination timetable will be discussed in §5.2.

After inspecting the entries of the *NAME OF PRECEDING* column and consulting the head of timetables at Stellenbosch University, it was clear that all of the entries in this column corresponds with modules that have more than one examination paper, so this column could also be deleted from the dataset. If a module with two examination papers would require Paper 1 to be written before Paper 2, and this constraint was ignored during the scheduling of the examination timetable so that Paper 2 is scheduled in an earlier timeslot than Paper 1, it can easily be rectified by swapping the positions of the two papers in the timetable. This will not cause any clashes for any student, because a student that is required to write Paper 1 must also write Paper 2. Thus, all of the other examination papers scheduled on the same timeslot as Paper 1 will cause no clashes in the timetable, so all of these examination papers will also not cause any clashes with Paper 2. Similarly will all the papers scheduled on the same timeslots as Paper 2 cause no clashes with either Papers 1 or 2.

The *OTHER REQUIREMENTS* column is also dropped from the datasets. This column is not necessary for a feasible solution, as it only contains information on lecturers' preferences. Although this column is dropped, a way of handling lecturers' preferences in the timetable is discussed in §5.2.

An example of an extract of one of the final datasets after these changes has been made, can be seen in Table 4.5.

NAME	FIXED DATES	NAME OF SAME TIME
Agronomie_454 Vr1		
AkadGelEBW_111 Vr1		
AktWet_112 Vr1	day 1, morning	
AktWet_142 Vr1	day 7, morning	
AktWet_242 Vr1	day 9, morning	
AktWet_274 Vr1	day 12, morning	FinRisBest 274
AktWet_274 Vr1	day 1, morning	
AktWet_326 Vr1	day 3, morning	
AktWet_346 Vr1	day 3, morning	
AktWet_388 Vr1	day 10, morning	
AktWet_388 Vr1	day 8, morning	
Akwakult_314 Vr1		
Akwakult_344 Vr1		
Akwakult_414 Vr1		
Akwakult_444 Vr1		

TABLE 4.5: *An example of the information on the examination papers after datahandling is done.*

4.1.4 Timetabling graphs of Stellenbosch University

After the required data handling as discussed so far in the section is done on the provided datasets, the method described in §3.1 is used to obtain a timetabling graph for Stellenbosch University.

In Table 4.6 the information on the datasets given by Stellenbosch University is displayed. Datasets 1 and 2 correspond to the first and second semester of 2016, whereas datasets 3 and 4 correspond to the first and second semester of 2017, respectively.

Dataset	# of modules	# of examination papers	# of enrolments	# of students
1	1 067	606	111 559	21 284
2	1 088	644	109 081	20 512
3	1 052	587	113 417	21 368
4	1 016	623	109 162	20 480

TABLE 4.6: *Information on the datasets from Stellenbosch University, where datasets 1 and 2 correspond to the first and second semester of 2016, and datasets 3 and 4 correspond to the first and second semester of 2017, respectively.*

The method to obtain a timetabling graph in §3.1 consists of identifying the vertices and edges of the graph, as well as merging vertices that correspond with examination papers that must be written during the same timeslot. The vertices of Stellenbosch University's examination timetable are obtained from the examination papers listed in the *NAME* column of the final timetable data provided by Stellenbosch University (see Table 4.3) in §4.1.2. As in §3.1, the edges can be identified in the *list of examination papers* from the final datasets constructed from the examination timetables (see Table 4.4) in §4.1.2 by adding the vertices that correspond with the students' examination papers to each other's neighbourhoods. Finally, the *NAME OF SAME TIME* column discussed in §4.1.3 is used to identify the vertices that must be merged and the *FIXED DATES* column is used to identify vertices that have fixed colours.

Information regarding the resulting timetabling graph from each of the datasets in Table 4.6 is given in Table 4.7. In Table 4.7, graph G_i is the timetabling graph of dataset i in Table 4.6.

Graph	$p(G_i)$	$q(G_i)$	Density	# of vertices with fixed colours	# of merged vertices	# of colours allowed
G_1	542	9 705	0.066	4	41	36
G_2	556	9 810	0.064	9	53	40
G_3	545	9 582	0.065	4	37	36
G_4	558	9 485	0.061	9	51	40

TABLE 4.7: *Information on the timetabling graphs, where G_i is the timetabling graph obtained from dataset i in Table 4.6.*

From Table 4.7 one may see that all four timetabling graphs have a similar density, indicating that the difficulty of colouring these graphs are similar. Furthermore, it can be noted that the number of vertices with fixed colours are the same for G_1 and G_3, as well as for G_2 and G_4. The reason for this is because Stellenbosch University have a program with fixed timeslots for the examination papers for some years now. Some of the examination papers in this program require special external moderation and these papers need to be scheduled in specific timeslots. From Tables 4.6 and 4.7 it is noted that the number of examination papers of a particular dataset i is not necessarily equal to the $p(G_i)$ due to the merged vertices that represent examination papers that must be scheduled during the same timeslot.

4.2 Parameter calibration

The parameters of the cost function described in §3.4, together with the parameters of Algorithm 3 and Algorithm 4, must be calibrated before the datasets may be applied. In §4.2.1, experimentation with the parameters of the cost function in order to get appropriate parameter values,

is discussed. The calibration of the parameters of Algorithm 3 and Algorithm 4 is discussed in §4.2.2 and §4.2.3 respectively. All test runs in this project were executed on a computer with a Core i7, 3.6GHz processor.

4.2.1 Cost function

The cost function used in this project is newly derived and thus the interaction between the parameters μ, δ and ρ of the cost function is unknown. Appropriate values for these parameters are estimated by means of experimentation.

Parameter μ in function (3.3) influences the severity of the penalty in the cost function for a spacing between any student's pair of consecutive examination papers smaller than the perfect spacing. The larger the value of μ, the more penalty is added. On the other hand, the function of parameter δ in function (3.3) is to improve the cost function if any spacing between any student's pair of consecutive examination papers is larger than the prefect spacing. A larger value for δ would result in a larger improvement of the cost function. The purpose of ρ in function (3.5) is to penalise the clusters of examination papers of students. Once again, the larger the value of ρ, the more penalty is added to the cost function.

In an attempt to find appropriate values for μ, δ and ρ, Algorithm 3 is run several times with different combinations of these three parameters in the cost function. The combinations of parameter values used during the test runs can be seen in Table 4.8.

Combination #	μ	δ	ρ
1	1.1	0	2
2	1.1	0.1	2
3	1	2	2
4	2	0.1	2
5	10	0.1	2
6	1.1	0.1	4

TABLE 4.8: *The different combination of the three parameters of the cost function.*

At this stage, appropriate values for parameters γ and R in Algorithm 3 are not determined yet. From experimentation, though, it is known that running Algorithm 3 for more than 100 000 iterations does not improve the cost function value significantly. These 100 000 iterations take approximately 16 hours to finish. Therefore, the termination parameter is taken as $\gamma = 100\,000$. The set of rules R was selected to be to choose a random move at every iteration for all experimental runs. All the experimental runs was applied to the same timetabling graph G_2, obtained in §4.1.4. Furthermore, Algorithm 3 is applied 10 times to G_2 for each of the parameter combinations.

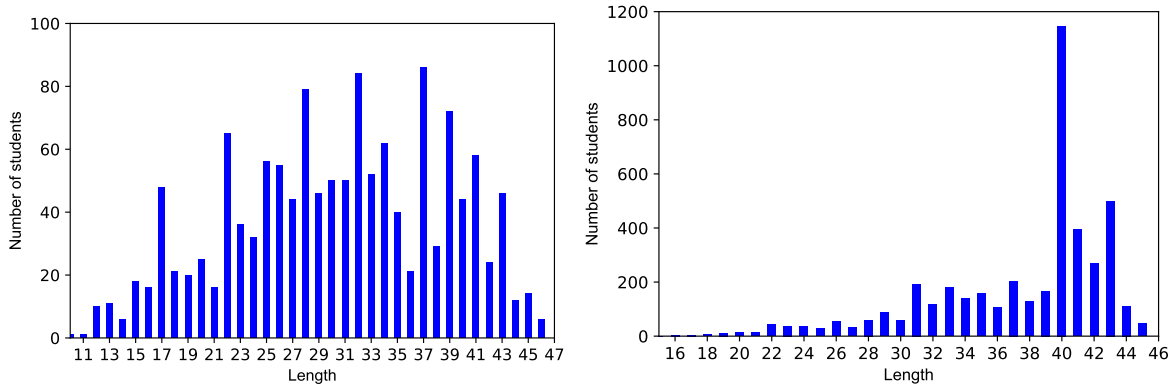
Students were catogorised according to how many examination papers they have to write. The number of students that have to write a certain number of examination papers can be seen in Table 4.9.

Examples of the two typical types of histograms of the number of students over the 10 runs who had the specific examination period lengths indicated, are given in Figure 4.1, where the duration of each students' examination period was taken as the number of timeslots from a student's first examination paper up and including the student's last examination paper. In both cases, parameter combination 4 (chosen arbitrarily) in Table 4.8, and $\gamma = 100\,000$ was used and in Figure Table 4.1(a) [(b), respectively] is the histogram of the number of students that are

Dataset	3 Papers	4 Papers	5 Papers	6 Papers	7 Papers	8 Papers	9 Papers	10 Papers	11 Papers	12 Papers
1	941	2 462	4 765	4 534	1 983	534	316	178	21	5
2	1 369	2 794	4 356	2 939	3 162	622	363	128	23	
3	969	3 379	4 503	4 607	1 847	479	400	53	11	
4	1 442	3 187	4 170	2 820	3 014	716	342	131	16	4

TABLE 4.9: *The number of students that will have to write the listed number of examination papers in each case, where datasets 1 and 2 correspond to the first and second semester of 2016, and datasets 3 and 4 correspond to the first and second semester of 2017, respectively.*

required to write 3 [5, respectively] examination papers. The histogram in Figure 4.1(a) seems like a normal curve around the examination period of length 30 and the average examination period length of students with 3 examination papers is 30.52 timeslots. From Table 4.9, it is clear that most students are required to write 5 examination papers and in the histogram in Figure 4.1(b), there is a large number of students that have an examination period of length 40 timeslots, so one would expect that the average will tend towards 40. The average is 39.01 timeslots.



(a) (b)

FIGURE 4.1: *Histograms of the number of students that had certain examination period lengths, where (a) is for students who are required to write 3 examination papers and (b) students who are required to write 5 examination papers. These results were obtained by executing Algorithm 3 ten time on timetabling graph G_2, with $\gamma = 100\,000$.*

Therefore, for each parameter combination in Table 4.8, the average duration of the examination period for each group of students with a certain number of papers, was calculated. The results are illustrated in Figure 4.2.

It can be seen from Figure 4.2 that, on average, the use of combination 3 resulted in students' examination periods being longer than when using any other combination of parameters, while the use of parameter combination 5 resulted in shorter spans of students' examination periods. In parameter combination 3, δ is relatively large compared to the other combinations, meaning that larger spacings between students' consecutive examination papers are rewarded more in the cost function than with the other parameter value combinations.

It is believed that the reason for the shorter examination periods for parameter combination 5, is because the difference between μ and δ is relatively large. With these values, spacings between consecutive examination papers that are smaller than the perfect spacing get penalised heavily,

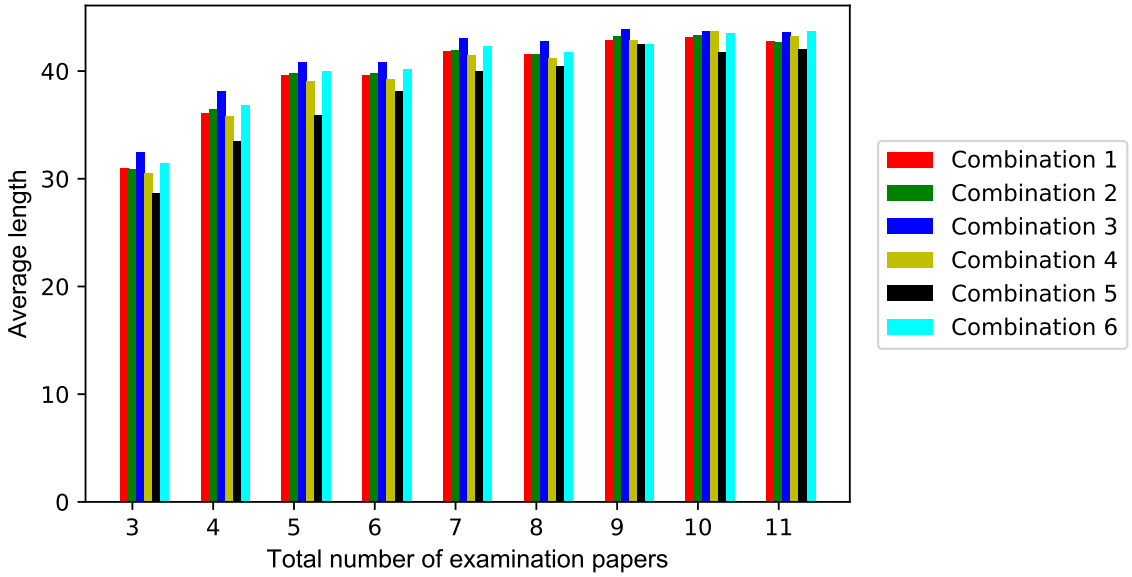


FIGURE 4.2: *The average length of a students' examination period, where students were categorised according to the number of examination papers they have to write. These results were obtained via Algorithm 3 on G_2, with $\gamma = 100\,000$ and selecting a random move at each iteration. The average lengths were computed over 10 executions of Algorithm 3. The combinations in the legend refers to the parameter combinations in Table 4.8.*

while spacings between consecutive examination papers that are too large are almost completely ignored. Thus, when using combination 5 in the cost function, the algorithm will most likely find a solution in which the spacing between students' consecutive examination papers strive towards the perfect spacing, rather than trying to make the spacings between consecutive examination papers as large as possible.

Next, it was attempted to get an idea of how well spaced out the students' examination papers are between their first and last examination papers. Again, the students were divided in groups according to the number of examination papers they have to write.

The spacing of students' examination papers was expressed as the percentage of the total number of spacings between consecutive examination papers that are within three timeslots of each other, to the total number of spacings between consecutive examination papers for all students in the particular group of students. For example, from Table 4.9 it can be seen that in dataset 2 there are 1 369 students with three examination papers. Thus, there are a total of $1369 \times (3-1) = 2\,738$ spacings between consecutive examination papers for all these students collectively. There are 10 runs of the algorithm for each combination of parameters values, resulting in a total number of spacings between consecutive examination papers over the 10 runs as $2\,738 \times 10 = 27\,380$. For parameter combination 1 in Table 4.8, for example, it was determined that there were 993 occurrences over all students in the group during the 10 runs that consecutive papers were within three timeslots of each other. This means that, on average, $\frac{933}{27\,380} \times 100 = 0.036\%$ of the total number of spacings between consecutive examination papers for students with three examination papers, are within three timeslots of each other. This was done for all parameter combinations in Table 4.8 on each set of students with a certain number of examination papers and the results are displayed in the graph in Figure 4.3.

From Figure 4.3 it is clear that the use of combination 3 as parameter choice for the cost function, resulted in the most number of consecutive examination papers that are within three timeslots

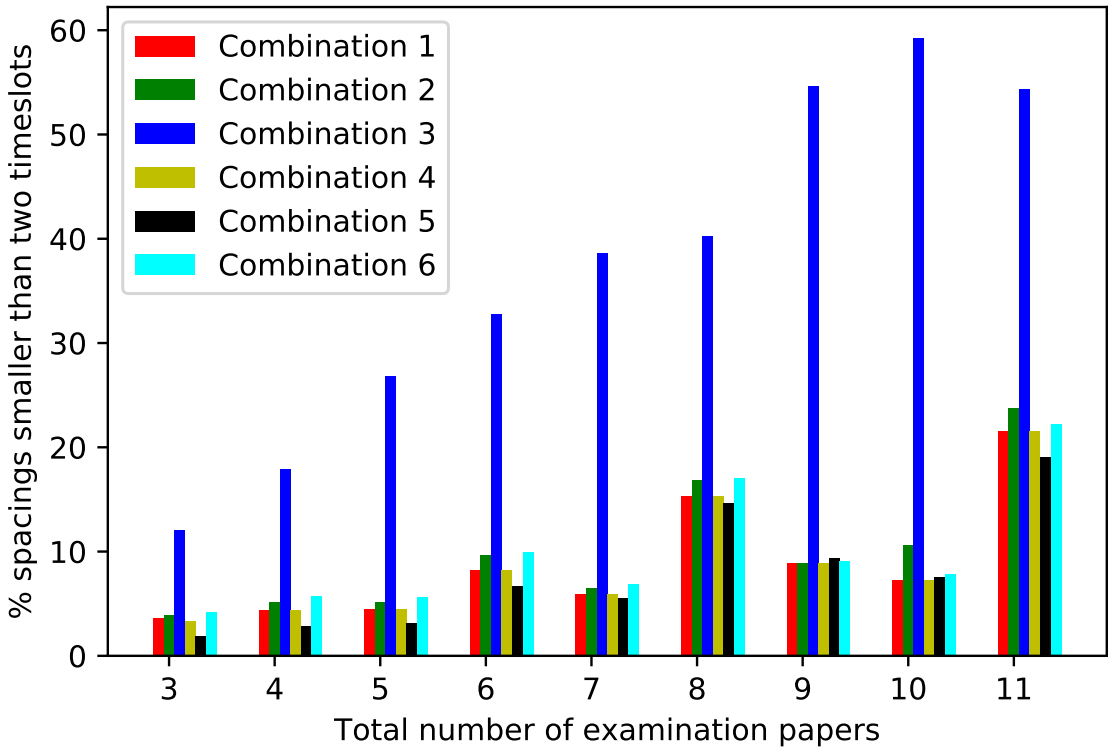


FIGURE 4.3: *The average percentage of consecutive examination papers written within three timeslots (one day) of each other, for students with a certain number of examination papers. These results were obtained by applying Algorithm 3 to G_2, with $\gamma = 100\,000$ and using a random move at each iteration. The average percentages were computed over 10 executions of Algorithm 3. The combinations in the legend refers to the parameter combinations in Table 4.8.*

of each other, being far worse than all the other parameter combinations. Contrarily, parameter combination 5 was, in most cases, the best when considering the number of examination papers being written within three timeslots of each other.

Comparing Figure 4.2 and Figure 4.3, one may select the appropriate combination of the parameter values for μ and δ in the cost function to obtain a good spacing between the students' examination papers over a large part of the examination period. For the purposes of this project, it was decided to use combination 4 as parameters for the cost function for the search algorithms, since the use of parameter combination 4 resulted in spreading an average student's papers over a relatively large part of the examination period, while keeping the number of consecutive examination papers scheduled within three timeslots of each other to a minimum.

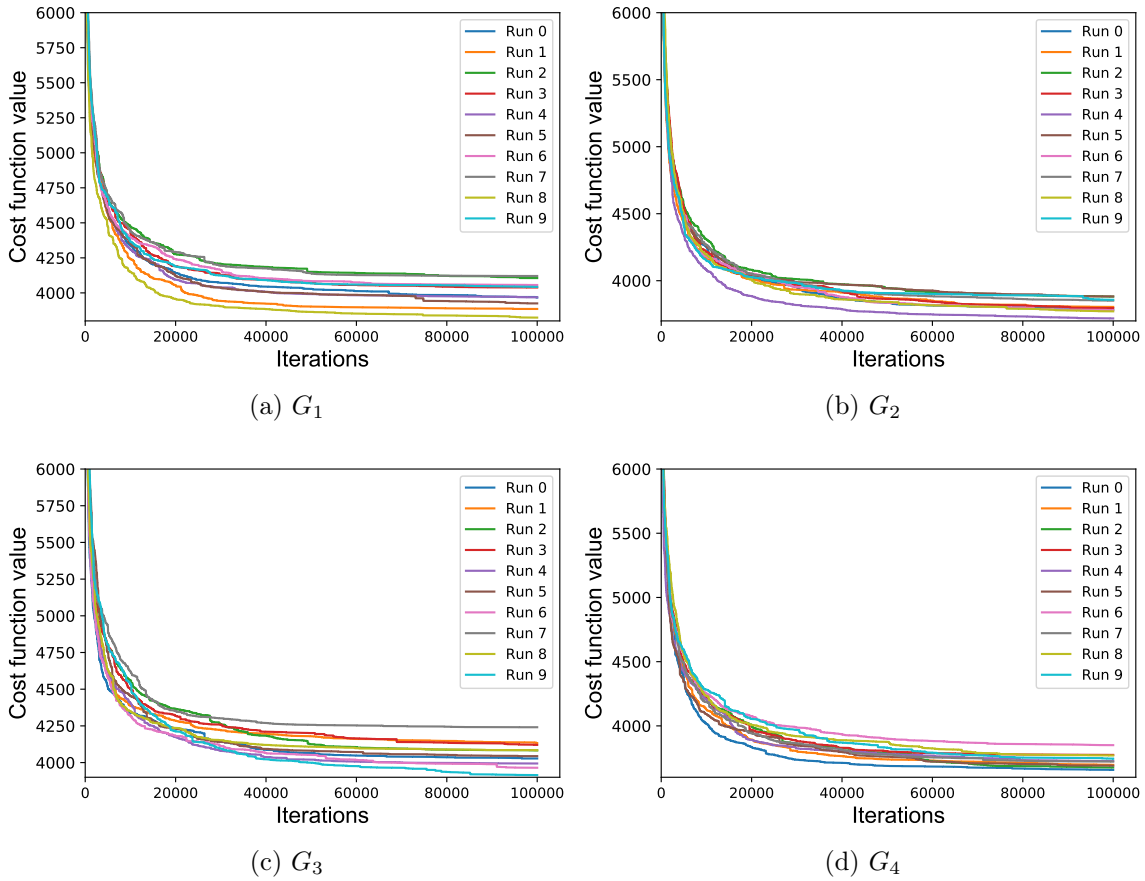
4.2.2 Hill climbing

The only parameters that need calibration in Algorithm 3, is R and γ. To determine a value for γ, the algorithm is executed with fixed R on the four timetabling graphs obtained in §4.1.4 for an excessive number of iterations ($\gamma = 100\,000$). After the algorithm is terminated, a graph of the cost function value against the number of iterations, is plotted. This should be done multiple times on multiple datasets. These graphs may then be inspected to see at which point little or no improvement on the cost function can still be obtained. The number of iterations at this point, when no or little improvement is made, may be used to estimate a value for parameter γ.

It should be noted that γ may change as R is changed, so γ should be calibrated for every R.

Different sets of rules R_i for choosing a move at each iteration is considered. Let R_1 consist of choosing a random move at each iteration, and R_2 be to iteratively use the move described in §3.3.1, followed by the move described in §3.3.2 and then the move of §3.3.3 before continuing with the first move again.

The cost function value against the number of iterations when Algorithm 3 is applied to the four timetabling graphs of §4.1.4, with $\gamma = 100\,000$ and R_1 and is given in Figure 4.4. A total number of 10 runs was executed on each of the four timetabling graphs with the same set of parameter values.



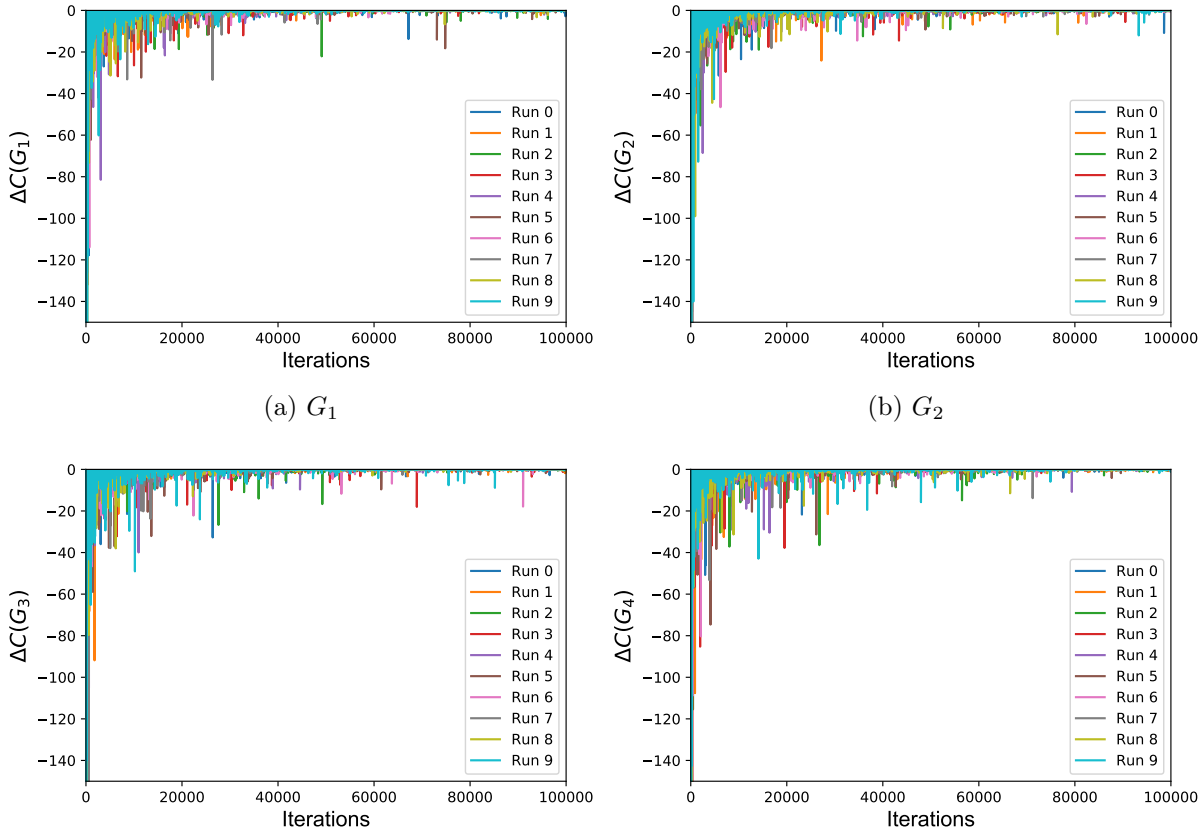
(a) G_1 (b) G_2

(c) G_3 (d) G_4

FIGURE 4.4: *Graphs of the cost function value over the number of iterations when Algorithm 3 is applied to the fourth timetabling graphs obtained in §4.1.4. The algorithm was set to terminate after* 100000 *iterations and the set of rules* R_1 *was used. The algorithm was executed* 10 *times on each timetabling graph.*

From roughly around 30 000 iterations the improvement in the cost function starts to decrease for all four timetabling graphs in Figure 4.4. The average values of the cost functions at 0, 30 000 and 100 000 iterations, can be seen in Table 4.10. From Table 4.10 one may see there was an improvement of, at most, 1.37% on the initial cost function value when Algorithm 3 was executed beyond 30 000 iterations to 100 000 iterations. It can be argued that a $\pm 1.37\%$ improvement on the cost function value is not enough to justify the extra ± 11 hour runtime needed after 30 000 iterations until 100 000 iterations. This claim can also be supported by studying the graphs given in Figure 4.5 of the change in the cost function values as the number of iterations increase. From these graphs it is apparent that there are almost no significant improvements to the cost

function after 30 000 iterations. Thus, a value of $\gamma = 30\,000$ is chosen when Algorithm 3 is used in combination with ruleset R_1.



(a) G_1 (b) G_2

(c) G_3 (d) G_4

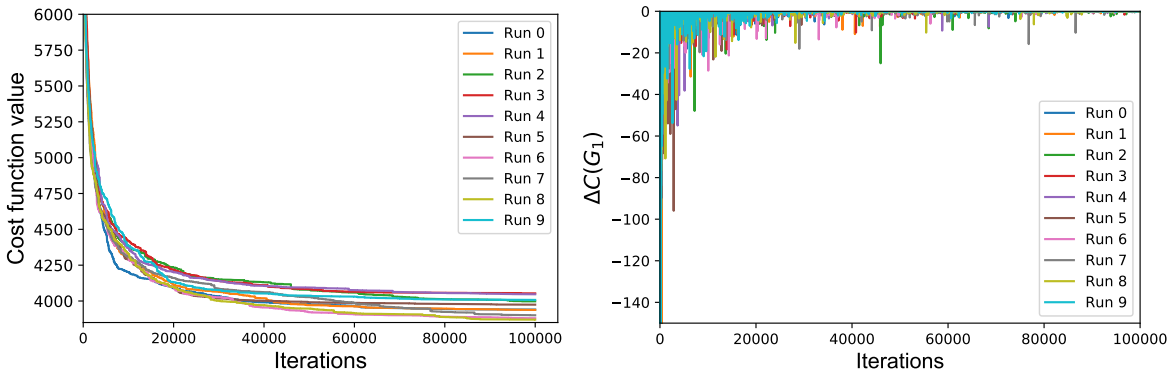
FIGURE 4.5: *Graphs of the change in the cost function value over the number of iterations when Algorithm 3 is applied to the fourth timetabling graphs obtained in §4.1.4. The algorithm was set to terminate after* 100 000 *iterations and the set of rules* R_1 *was used. The algorithm was executed* 10 *times on each timetabling graph.*

Graph	$C(G_i)$ at iteration 0	$C(G_i)$ at iteration 30 000	$C(G_i)$ at iteration 100 000	% lost with $\gamma = 30\,000$
G_1	10 210.09	4 081.43	3 993.33	0.86%
G_2	10 886.86	3 948.95	3 812.58	1.25%
G_3	11 118.77	4 176.61	4 060.63	1.04%
G_4	10 093.65	3 869.29	3 730.51	1.37%

TABLE 4.10: *The average cost function values at a certain number of iterations. These averages are calculated over* 10 *runs of Algorithm 3 with ruleset* R_1*. The last column shows the average percentage improvement on the initial cost function value that is lost when the algorithm terminates at* 30 000 *iterations, rather than at* 100 000 *iterations.*

Next, ruleset R_2 was used in Algorithm 3 and again the parameter was set as $\gamma = 100\,000$. The value of the cost function when Algorithm 3 is applied 10 times to the timetabling graph G_1 is given in Figure 4.6(a), while the change in the cost function value for the same runs is given in Figure 4.6(b). The graphs for the other three timetabling graphs may be found in Appendix A.

Once again, it can be observed that the cost function value only improves slightly after 30 000 iterations.



(a) (b)

FIGURE 4.6: *The graphs of Algorithm 3 applied to timetabling graph G_1 using the set of rules R_2, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after* 100 000 *iterations and was executed* 10 *times on timetabling graph G_1.*

As with the runs of Algorithm 3 with R_1, a summary of the improvement of the cost function value against the number of iterations is given for the runs of Algorithm 3 with R_2. This summary is given in Table 4.11. As before, from Table 4.11 it is clear that the initial cost function value will improve by, at most, 1.27% when Algorithm 3 is executed beyond 30 000 iterations. Therefore as with R_1, $\gamma = 30\,000$ will be used in future results when Algorithm 3 is used in combination with ruleset R_2.

Graph	$C(G_i)$ at iteration 0	$C(G_i)$ at iteration 30 000	$C(G_i)$ at iteration 100 000	% lost with $\gamma = 30\,000$
G_1	10 210.09	4 073.61	3 960.91	1.10%
G_2	10 886.86	4 064.36	3 935.28	1.19%
G_3	11 118.77	4 144.49	4 041.65	0.93%
G_4	10 093.65	3 840.67	3 712.24	1.27%

TABLE 4.11: *Table showing the average cost function values at a certain number of iterations. These averages are computed over* 10 *runs of Algorithm 3 with ruleset R_2.*

With an appropriate value for γ determined, the two rulesets are compared. Table 4.12 contains the average cost function values after 30 000 iterations when Algorithm 3 with R_1 and R_2, respectively, is applied to the four timetabling graphs. No clear conclusion can be made from the values in Table 4.12, on which ruleset performs better, since using R_1 resulted in a lower cost function value for G_2 than R_2, while R_2 had a better value for $C(G_1)$, $C(G_3)$ and $C(G_4)$ than when R_1 was used.

As mentioned in §3.3.2, it is expected that using the move of swapping the vertices of two colour classes should drastically change the cost function value. This large change in the cost function value might be beneficial at early stages of the execution of the algorithm, but it is expected that the algorithm will reject neighbours generated by this move at later stages, because the solution will strive to get closer to a local optimum at later iterations, meaning that a big change in the solution might increase the cost function as the search is moving away from the local optimum.

Graph	$C(G_i)$ with R_1	$C(G_i)$ with R_2
G_1	4 081.43	4 073.61
G_2	3 948.95	4 064.36
G_3	4 176.61	4 144.49
G_4	3 869.29	3 840.67

TABLE 4.12: *Table showing the average cost function values when R_1 and R_2, respectively, was used in Algorithm 3 with $\gamma = 30\,000$. These averages are computed over 10 runs of Algorithm 3.*

This expectation was investigated by executing Algorithm 3 on G_1, with $\gamma = 100\,000$ and R_2, and by recording whenever the move described in §3.3.2 resulted in an improvement of the cost function. This was repeated 10 times, and the results are given in Table 4.13, where the last five iterations when the use of the move resulted in an decrease of the cost function, were recorded.

Run #	Last iteration	Second last iteration	Third last iteration	Fourth last iteration	Fifth last iteration
1	**5 284**	**5 200**	**4 117**	**3 787**	2 458
2	2 689	1 876	1 330	853	643
3	**92 395**	**8 959**	892	538	487
4	**13 324**	**3 862**	1 969	1 393	1 228
5	**4 855**	**3 298**	634	325	250
6	**16 447**	**11 851**	**9 109**	8 581	2 891
7	**15 238**	**5 335**	2 542	2 501	2 032
8	**55 462**	**38 488**	2 581	2 146	1 399
9	**36 925**	**26 128**	**11 608**	1 777	1 474
10	**12 775**	**4 261**	**4 141**	1 747	1 435

TABLE 4.13: *Results obtained by executing Algorithm 3 on G_1, with $\gamma = 100\,000$ and ruleset R_2, where the last five iterations at which the move explained in §3.3.2 made an improvement on the current solution, were recorded.*

It can be seen in Table 4.13 that after 3 000 iterations, the move described in §3.3.2 rarely generates a neighbour that decreases the cost function. In fact, over the 10 runs, there were only 23 instances, indicated in bold in Table 4.13, where the move resulted in a decrease in the cost function after 3 000 iterations. If one considers that there are $100\,000 - 3\,000 = 97\,000$ iterations after 3 000 iterations was executed during each run of the algorithm, that the algorithm was executed 10 times, and that for R_2 the move at hand was used only every third iteration, this move was used a total of $\frac{97\,000}{3} \times 10 \approx 323\,330$ times after 3 000 iterations have already been executed over the 10 runs to generate a new neighbour. Thus, on average, this move generated a neighbour solution with a lower cost function value only $\frac{23}{323\,330} \times 100 = 0.0071\%$ of the time. Therefore, a new set of rules, R_3, is formed. This new set of rules is exactly the same as R_2, except that the move described in §3.3.2 will not be considered after 3 000 iterations have been executed. The resulting graphs when Algorithm 3 was executed 10 times on the four timetabling graphs with $\gamma = 30\,000$ and ruleset R_3, can be seen in Appendix A (Figure A.4).

Table 4.14 shows the average cost function values for all three rulesets after 30 000 iterations of Algorithm 3 was executed on the different timetabling graphs. It can be seen from this table that the use of R_3 resulted in the best average cost function value for three of the four timetabling graphs. The average runtime of Algorithm 3, with parameter values $\gamma = 30\,000$ and ruleset R_3, was 5.26 hours.

Graph	$C(G_i)$ with R_1	$C(G_i)$ with R_2	$C(G_i)$ with R_3
G_1	4 081.43	4 073.61	4 069.99
G_2	3 948.95	4 064.36	4 008.12
G_3	4 176.61	4 144.49	4 140.08
G_4	3 869.29	3 840.67	3 820.45

TABLE 4.14: *Table showing the average cost function values when using R_1, R_2 and R_3, respectively, in Algorithm 3 with $\gamma = 30\,000$. These averages are computed over 10 runs of Algorithm 3 on each of the timetabling graphs.*

4.2.3 The great deluge algorithm

The parameters used in Algorithm 4 is R, γ and θ, the acceptance level. After a brief experimentation, it was clear that Algorithm 4 has the same property as Algorithm 3 with regards to the move described in §3.3.2 in that the move rarely cause an improvement in the cost function after 3 000 iterations are executed. Therefore ruleset R_3 is also used for Algorithm 4.

Since $\gamma = 30\,000$ was a good estimated value to use in Algorithm 3, it was decided to execute Algorithm 4 only a few iterations longer to determine whether Algorithm 4 obtained significant improvements after 3 000 iterations. Thus, γ was set to 50 000 in the experimentations to follow.

The only parameters left to calibrate, are the acceptance level θ and the function $D(\theta)$, where $D(\theta)$ is used to lower θ whenever a neighbour solution is accepted as being the current solution. It was stated in §3.5.2 that Dueck's [23] choices of θ and $D(\theta)$ will be used in this project. Thus, θ will be equal to the cost function value of the initial solution and (3.7) will be used for $D(\theta)$. To calibrate β in $D(\theta)$, Algorithm 4 is executed 10 times on timetabling graph G_2 (chosen arbitrarily), with parameter settings $\gamma = 30\,000$, $\theta = C(initial\_solution)$ and R_3, for different values of β in $D(\theta) = \theta - \frac{\theta - C(neighbour\_solution)}{\beta}$. Table 4.15 contains the results of executing these runs for different values of β over 30 000 and 50 000 iterations[1], and the graphs of the cost function value over the iterations is given in Appendix B.

β value	$C(G_2)$ at 30 000 iterations	$C(G_2)$ at 50 000 iterations
1.5	3 921.48	3 843.76
2	3 967.66	3 893.42
3	3 945.76	3 863.67
4	3 962.99	3 869.12
10	3 957.97	3 807.92

TABLE 4.15: *The average cost function values after 30 000 and after 50 000 iterations of Algorithm 4 was executed, using ruletset R_3. These averages are computed over 10 runs of Algorithm 4 on timetabling graph G_2.*

It can be seen from Table 4.15 that using $\beta = 1.5$, the best results are obtained after 30 000 iterations, while using $\beta = 10$, the best results are obtained when the algorithm runs until 50 000 iterations have been reached. A smaller β value results in a steeper decline of acceptance level, while a larger β value causes the acceptance level to slowly decline. This could explain why the choice of $\beta = 1.5$ performs better at a smaller number of iterations. However, the steep decline of the acceptance level can make escaping a local optima more difficult at later stages of the

[1]Note that in Table 4.15, $\beta = 1$ was not considered, since $\beta = 1$ will always lower the acceptance level θ to the cost function value of the new solution, thus making it exactly the same as a hillclimbing algorithm.

algorithm. This might explain why $\beta = 10$ performed better than all the other β values after 50 000 iterations were executed, as the algorithm can move more easily when in a local optima. It is expected that using a large value for β over a long period of time ($\gamma = 100\,000$) would result in significantly better cost function values than using a small β over a long period of time[2].

4.3 Results

In this section, the results obtained via the various algorithms used in this project is discussed. First, the performance of Algorithm 1 together with Algorithm 2 to generate an initial solution is discussed in §4.3.1. Next, the results obtained via Algorithm 3 and Algorithm 4 are compared in §4.3.2. The topic of §4.3.3 is to compare the performance of the cost function used in this project with a cost function from the literature.

4.3.1 Initial solution

The 2–phase approach of using Algorithm 1's output as input for Algorithm 2 to generate a feasible initial solution of a timetabling graph, was tested on all of the datasets that were provided by Stellenbosch University.

During this project, the 2–phase approach never failed to generate a feasible initial solution for any given timetabling graph. Furthermore, this approach could even generate feasible initial solutions that have less colours than what was expected for a feasible solution. In Table 4.16 the number of colours that were allowed for the different timetabling graphs used in this project, together with the minimum number of colours that the 2–phase approach could use as input and still generate a feasible initial solution, are displayed. From the results in Table 4.16, it can be observed that the 2–phase approach can even be used to find a feasible initial solution of the second opportunity examination period[3] which is not included in the scope of this project.

Graph	# colours required	minimum # colours
G_1	36	29
G_2	40	30
G_3	36	29
G_4	40	28

TABLE 4.16: *The minimum number of colours that the 2-phase approach, using Algorithm 1's output as input for Algorithm 2, could use to generate a feasible initial solution for all the timetabling graphs.*

4.3.2 The performance of Algorithm 3 and Algorithm 4

Comparing the results of Algorithm 4 in Table 4.15 with the results of Algorithm 3 in Table 4.14, one notice that using $\beta = 1.5$ in Algorithm 4 resulted in an average cost function value of 3 921.48 after 30 000 iterations, while Algorithm 3 resulted in a worse average cost function of 3 948.95 for timetabling graph G_2. Algorithm 4 thus outperforms Algorithm 3 at 30 000 iterations, but not by much.

[2]This is almost the same as simulated annealing with a slow cooling function delivering good results over a long time period. It is expected that using a fast cooling function delivers better results in the short term, though.

[3]The second opportunity period lasts for 15 days, in other words 30 timeslots.

When the average cost function value of 3 935.28 in Table 4.11 after 100 000 iterations of Algorithm 3 applied to timetabling graph G_2 is compared to the average cost function values after 50 000 iterations of Algorithm 4 in Table 4.15, it should be clear that Algorithm 4 outperforms Algorithm 3, as any choice of β resulted in a better average cost function value than 3 935.28 in half the execution time.

4.3.3 Comparing the results with literature

In this section, the cost function of the 2nd International Timetabling Competition (ITC), as briefly explained in §3.4.1, is compared to the cost function used in this project. For more information on the cost function used in the ITC, and how this cost function was adapted to fit the variant of the ETP used in this project for comparison purposes, see Appendix C.

The limitation, as discussed in §3.4.1, of the ITC's cost function, was that it did not take the number of examination papers a particular student needs to write, into account when the examination papers for this student is spread out. The newly derived cost function of this project focused on spreading out students' examination papers based on the number of examination papers the student has to write.

To compare the newly derived cost function with the cost function of the ITC, the two cost functions were used in Algorithm 3 with parameter settings R_2 and $\gamma = 100\,000$[4] on timetabling graph G_2. The same tests that were done in §4.2.1 to investigate the length of students' examination periods and the number of times students have to write consecutive examination papers within 3 timeslots of each other, was applied to the solutions that were obtained from using the different cost functions. The resulting graphs are displayed in Figure 4.7 and Figure 4.8.

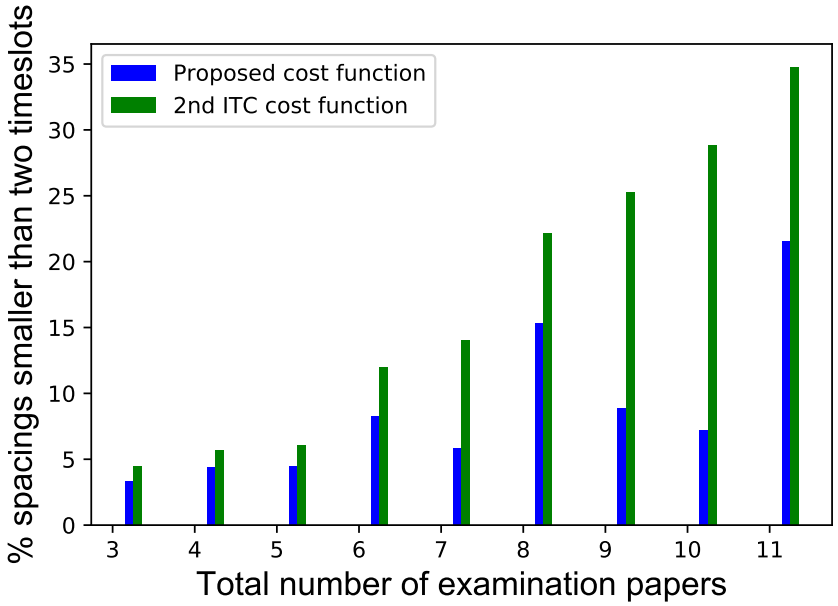


FIGURE 4.7: *The average percentage of consecutive examination papers written within one day of each other for a students with a certain number of examination papers. These results were obtained via Algorithm 3 on G_2, with $\gamma = 100\,000$ and R_2, using the proposed cost function of this project and the cost function of the ITC.*

[4] An excessive number of iterations were used, to ensure that the solutions are close to optimal.

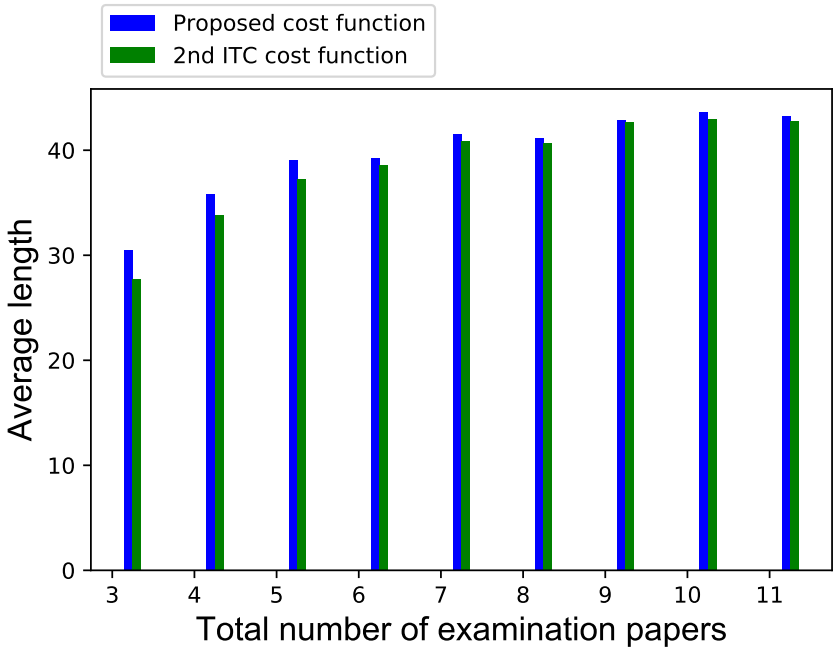


FIGURE 4.8: *The average length of a students' examination period, where students were categorised according to the number of examination papers they have to write. These results were obtained via Algorithm 4 on G_2, with $\gamma = 100\,000$ and R_2, using the proposed cost function of this project and the cost function of the ITC.*

It can be seen from Figure 4.7 that the use of the proposed cost function of this project resulted in less students having to write consecutive examination papers within 3 timeslot of one another, compared to using the cost function of the ITC. This, on average, is true for all students, no matter how many examination papers they had to write. Furthermore, using the proposed cost function outperformed the cost function of the ITC by a large margin when the number of consecutive papers for students with more than 5 examination papers have to write within 3 timeslots.

From Figure 4.8, it can also be seen that the use of the proposed cost function outperforms the cost function of the ITC in terms of the length of students' examination periods. This is especially notable for students with a small number (3,4 or 5) of examination papers.

CHAPTER 5

Conclusion

Contents

"A conclusion is simply the place where you got tired of thinking. "

Dan Chaon (2012)

A summary of this project is discussed in §5.1. The chapter is the concluded in §5.2 with ideas for future work that could build on the findings in this project.

5.1 Summary

The purpose of this study was to do research on the possibility of an examination timetabling method that will spread most students' examination papers equally over the duration of the examination period. Stellenbosch University is used as a case study. Graph colouring together with search algorithms are used to solve this problem, and the relative literature is discussed in Chapter 2. The solution approach is explained in Chapter 3, where the process of obtaining timetabling graphs from the datasets provided by Stellenbosch University is described in §3.1, which addresses Objective I. A two–phase approach of finding an initial solution satisfying all the hard constraints at Stellenbosch University, as to satisfy Objective II, is explained in §3.2. The first phase consists of finding a proper colouring of the timetabling graph, after which an equitable colouring is sought in the second phase. The objective function used in this project is a cost function that reflects how well the students' examination papers are spaced out within a given examination timetable. This cost function is discussed in §3.4, thus addressing Objective III. The cost function is derived in such a way as to make it fair towards all students, no matter how many examination papers they are required to write. It is found that this cost function performs better than the cost function used at the 2nd International Timetabling Competition in terms of spacing students' examination papers over the entire examination period. Two search algorithms, namely hill climbing and the great deluge algorithm as discussed in §3.5, are used to search through the solution space for better timetable solutions. These algorithms make use

of three simple moves, discussed in §3.3, to generate neighbour solutions of the current solution. It is found that using the moves in a set order performs better than choosing a random move at every iteration. The moves, together with the cost function, are used in the search algorithms to satisfies Objective IV. After parameter calibration is done on the parameters of the algorithms and the cost function in §4.2, it is found that the great deluge algorithm outperforms hill climbing. This addresses Objective V. Some hard and soft contraints of Stellenbosch University's variant of the examination timetabling problem were not in the scope of this project, due to the unavailability of data. Objective VI will be addressed in §5.2 by explaining possible ideas of incorporating these constraints into the algorithms of this project and discussing future research on how to improve the algorithms.

5.2 Future work

Possible ideas and future research that could build upon the finding of this project, is discussed in this section. The first topic describes how the algorithms in this project will need to be adjusted in order for Stellenbosch University to implement it, since some hard and soft constraints were not in the scope of this project.

Adjusting the algorithms

There are some soft and hard constraints used by Stellenbosch University that was not incorporated in this project. The hard constraints consist of examination papers that must/may not be scheduled during certain time frames of the examination timetable, while the soft constraints consist of lecturers' preferences as to when they want their examination papers to be scheduled.

Changing the algorithms described in this project to accommodate the missing hard constraints, is to assign a list containing all possible colours that the vertex may be to every vertex in the timetabling graph. For example, if a examination paper must be scheduled during the first 11 timeslots, the vertex representing that examination paper will have a list containing only those 11 colours. Furthermore, if a examination paper must not be scheduled during the last 5 timeslots, the vertex representing that examination paper will have a list containing all of the available colours, except for the 5 colours representing these 5 last timeslots. At the moment, the algorithms described in this project only considers vertices that must be scheduled during a specific timeslot. Thus the current algorithms assume that these lists, showing which colours are allowed per vertex, contain either all of the available colours, or only one colour. The only algorithm that will need to be adjusted to accommodate this change is Algorithm 1 and the three moves used to generate neighbours.

The change needed in Algorithm 1 is in Step 2, where the difficulty $f(u)$ to colour a vertex u will have to change to $f(u) = |\mathcal{T}_u| - \varrho(u)$, where $\mathcal{T}_u$ is the list of colours that is allowed to be assigned to vertex u and $\varrho(u)$ is the saturation degree of vertex u. In the first move described in §3.3.1, the set $\mathcal{A}_u$ must be updated so that $\mathcal{A}_u$ may only contain colours within $\mathcal{T}_u$.

The second move, described in §3.3.2, might have to be discarded after the new hard constraint is incorporated. The reason for this is that swapping the vertices of two colour classes might prove to be difficult if the vertices have an extra restriction on the colours that may be assigned to them. It might be nearly impossible to find two colour classes that may be swapped without breaking feasibility. It would, however, be possible to adjust this move to search for two such colour classes, but it is expected that the resulting move will perform poorly.

Finally, the move described in §3.3.3 to swap the timeslots of two examination papers may also be adjusted. Let vertex u, with colour m, and vertex v, with colour l, correspond to two examination papers that need to be swapped. The timeslots of these two examination papers will only be considered to be swapped if $\{l\} \in \mathcal{T}_u$ *and* $\{m\} \in \mathcal{T}_v$.

The soft constraints that were not in the scope of this project, can be incorporated by adding an extra term in the cost function of this project. This extra term, $P(G)$, is a function to penalise examination papers within timetable G that are not scheduled according to the lecturer's preference. The resulting cost function becomes

$$C(G) = \phi \left(\sum_{i=1}^{t} \frac{T(i)}{T_{i,w}} + \frac{K(i)}{K_{i,w}} \right) + (1 - \phi)P(G), \qquad \{\phi \in \mathbb{R} | 0 \leq \phi \leq 1\} \tag{5.1}$$

To be able to use equation (5.1) as cost function, data on the lecturers' preferences will have to be obtained. Using this data, a list can be made for every vertex of the timetabling graph that contains information on how preferable it would be for the vertex to have certain colours. These lists can be used in function $P(G)$ of cost function (5.1) to generate a penalty to examination papers that are not scheduled according to lecturers' preferences. Furthermore, the term ϕ in this cost function gives the user an option to choose how important they rate spacing out students' examination papers against lecturers' preferences of when their examination papers should be scheduled. Larger values of ϕ would result in students' examination papers to be prioritised more than pleasing lecturers.

Space available in venues

An assumption was made in §1.3 that as long as the colouring of a timetabling graph of Stellenbosch University is an equitable colouring, there will be enough room available in the venues to accommodate all students during each timeslot. This assumption might be accurate for the near future, but this might cause problems if Stellenbosch University expands in the sense of the number of students enrolled at the university, or even if some venues become unavailable for use during the examination period. To accommodate this in the future, the algorithms described in this project may have to be adjusted for future used.

Since allocating venues to examination papers was not in the scope of this project, some major adjustments will have to be made in the algorithms used in this project. Data about the venues available for use will also have to be provided. Using this data, it is possible to check in each iteration of the search algorithm if the generated neighbour solution satisfies the space constraint. This can be done, for example, by solving a **packing problem** for the neighbour solution to try and fit all of the students during every timeslot into available venues.

Using previous year's list of enrolments

Stellenbosch University currently use the previous years list of student enrolments to generate an examination timetable. The head of timetables compensate for new module choices for the current year by creating a 'dummy' student who has these new module choices, and then adding it to the previous year's list of enrolments. This process ensures a clash free timetable to be generated.

It is unknown what the consequences will be, in terms of spacing out students' examination papers, if the previous year's list of enrolments is used to generate an examination timetable.

This can be investigated by creating two timetables for every semester with the methods described in this project. The one timetable must be generated by using the previous year's list of enrolments, and the other one must be obtained by using the current year's list of student enrolments. These two timetables can then be compared to see if there are any consequences when the previous year's list of enrolments is used to generate an examination timetable.

It is expected that the spacing of students' examination papers will be worse when the previous year's list of enrolments is used. The reason for this, is because the resulting timetabling graph, formed from the old list of enrolments, will likely have unnecessary edges. These extra edges might form because of some module combinations not being available as choices for the next year, or simply because there are no students with those module combinations. Having extra edges in the timetabling graph causes unnecessary restrictions to be placed on the timetable, which could result in students' examination papers not being spaced out as much as possible.

Furthermore, it is expected that using a dummy student to compensate for a new module combinations will result in students' examination papers not being spaced out as best as possible. Only one dummy student is added per combination of modules. This does not give accurate data as to how many students are actually enrolled for those module combinations, and this data is vital in spacing out students' examination papers.

Improved moves

The moves used to generate neighbour solutions in this project, is fairly simple. It is worth investigating whether more complicated moves in conjunction with the search algorithms described in this project will improve the cost function value.

Researchers [24, 35, 43] have found that the use of Kempe chains to generate neighbouring solutions from the current solution performed exceptionally. Future work could consist of analising the use of Kempe chains in the moves used in this variant of the ETP to determine whether it will influence the resulting examination timetables of Stellenbosch University.

Improved parameter calibrations

The great deluge algorithm used in this project can also be improved by allowing θ to 're-heat'. McMullan [33] used this idea of re–heating θ (similar to simulated annealing) after a certain number of iterations have passed without finding an improved solution. He used a $D(\theta)$ that decreases θ significantly during the first 50% of the overall runtime, whereafter the use of $D(\theta)$ results in a small change in θ.

Bibliography

[1] Albertson MO and Moore EH, 1999, *Extending graph colorings*, Journal of Combinatorial Theory, Series B, **77(1)**, pp. 83–95.

[2] Barry McCollum, *International Timetabling Competition*, Available from `http://www.cs.qub.ac.uk/itc2007/examtrack/initialdatasets/exam_comp_set1.exam`.

[3] Battistutta M, Schaerf A and Urli T, 2014, *Feature-based tuning of single-stage simulated annealing for examination timetabling*, Annals of Operations Research, pp. 1–16.

[4] Battiti R, Bertossi A and Cavallaro D, 2001, *A randomized saturation degree heuristic for channel assignment in cellular radio networks*, IEEE Transactions on Vehicular Technology, **50(2)**, pp. 364–374.

[5] Bianchi L, Dorigo M, Gambardella LM and Gutjahr WJ, 2009, *A survey on metaheuristics for stochastic combinatorial optimization*, Natural Computing, **8(2)**, pp. 239–287.

[6] Bondy JA, Murty USR *et al.*, 1976, *Graph theory with applications*, volume 290, Citeseer.

[7] Brélaz D, 1979, *New methods to color the vertices of a graph*, Communications of the ACM, **22(4)**, pp. 251–256.

[8] Brelaz D, Nicolier Y and de Werra D, 1977, *Compactness and balancing in scheduling*, Zeitschrift für Operations Research, **21(1)**, pp. 65–73.

[9] Bullnheimer B, 1997, *An examination scheduling model to maximize studentsâĂŹ study time*, Proceedings of the International Conference on the Practice and Theory of Automated Timetabling, pp. 78–91.

[10] Burke E, Bykov Y, Newall J and Petrović S, 2003, *A time-predefined approach to course timetabling*, Yugoslav Journal of Operations Research, **13(2)**, pp. 139–151.

[11] Burke E, Bykov Y, Newall J and Petrovic S, 2004, *A time-predefined local search approach to exam timetabling problems*, Iie Transactions, **36(6)**, pp. 509–528.

[12] Burke E, Elliman D, Ford P and Weare R, 1995, *Examination timetabling in british universities: A survey*, pp. 76–90 in *Practice and Theory of Automated Timetabling*, pp. 76–90. Springer.

[13] Burke E, Jackson K, Kingston JH and Weare R, 1997, *Automated university timetabling: The state of the art*, The computer journal, **40(9)**, pp. 565–571.

[14] Burke EK, Newall J and Weare R, 1998, *A simple heuristically guided search for the timetable problem*, Proceedings of the Proceedings of the international ICSC symposium on engineering of intelligent systems (EIS98), pp. 574–579.

[15] Burke EK and Newall JP, 2004, *Solving examination timetabling problems through adaption of heuristic orderings*, Annals of operations Research, **129(1-4)**, pp. 107–134.

[16] Carrington JR, Pham N, Qu R and Yellen J, 2007, *An enhanced weighted graph model for examination/course timetabling*, Proceedings of the Proceedings of the 26th workshop of the UK planning and scheduling special interest group, pp. 9–16.

[17] Carter MW, 1986, *Or practiceâĂŤa survey of practical applications of examination timetabling algorithms*, Operations research, **34(2)**, pp. 193–202.

[18] Carter MW and Johnson D, 2001, *Extended clique initialisation in examination timetabling*, Journal of the operational research society, pp. 538–544.

[19] Carter MW, Laporte G and Lee SY, 1996, *Examination timetabling: Algorithmic strategies and applications*, Journal of the Operational Research Society, pp. 373–383.

[20] Daskalaki S and Birbas T, 2005, *Efficient solutions for a university timetabling problem through integer programming*, European Journal of Operational Research, **160(1)**, pp. 106–120.

[21] Di Gaspero L and Schaerf A, 2000, *Tabu search techniques for examination timetabling*, Proceedings of the International Conference on the Practice and Theory of Automated Timetabling, pp. 104–117.

[22] Dimopoulou M and Miliotis P, 2001, *Implementation of a university course and examination timetabling system*, European Journal of Operational Research, **130(1)**, pp. 202–213.

[23] Dueck G, 1993, *New optimization heuristics: The great deluge algorithm and the record-to-record travel*, Journal of Computational physics, **104(1)**, pp. 86–92.

[24] Duong TA and Lam KH, 2004, *Combining constraint programming and simulated annealing on university exam timetabling.*, Proceedings of the RIVF, pp. 205–210.

[25] Franken S, 2017, Head of timetables at Stellenbosch University, Personal communication.

[26] Hansen P, Hertz A and Kuplinsky J, 1993, *Bounded vertex colorings of graphs*, Discrete Mathematics, **111(1-3)**, pp. 305–312.

[27] Kierstead HA and Kostochka AV, 2008, *A short proof of the hajnal–szemerédi theorem on equitable colouring*, Combinatorics, Probability and Computing, **17(2)**, pp. 265–270.

[28] Leighton FT, 1979, *A graph coloring algorithm for large scheduling problems*, Journal of research of the national bureau of standards, **84(6)**, pp. 489–506.

[29] Marx D, 2004, *Graph colouring problems and their applications in scheduling*, Periodica Polytechnica Electrical Engineering, **48(1-2)**, pp. 11–16.

[30] McCollum B, McMullan P, Burke EK, Parkes AJ and Qu R, 2007, (Unpublished) , Technical report, Technical Report QUB/IEEE/Tech/ITC2007/-Exam/v4. 0/17, QueenâĂŹs University, Belfast.

[31] McCollum B, McMullan P, Parkes AJ, Burke EK and Abdullah S, 2009, *An extended great deluge approach to the examination timetabling problem*, Proceedings of the 4th multidisciplinary international scheduling: Theory and applications 2009 (MISTA 2009), pp. 424–434.

[32] McCollum B, McMullan P, Parkes AJ, Burke EK and Qu R, 2012, *A new model for automated examination timetabling*, Annals of Operations Research, **194(1)**, pp. 291–315.

[33] Mcmullan P, 2007, *An extended implementation of the great deluge algorithm for course timetabling*, Proceedings of the International Conference on Computational Science, pp. 538–545.

[34] Mehta NK, 1981, *The application of a graph coloring method to an examination scheduling problem*, Interfaces, **11(5)**, pp. 57–65.

[35] Merlot LT, Boland N, Hughes BD and Stuckey PJ, 2002, *A hybrid algorithm for the examination timetabling problem*, Proceedings of the International Conference on the Practice and Theory of Automated Timetabling, pp. 207–231.

[36] Nieuwoudt I, 2007, *On the maximum degree chromatic number of a graph*, Doctoral Dissertation, Stellenbosch: Stellenbosch University.

[37] Oracle, *Oracle SQL Developer 4.1.1*, Available from `http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/sqldev-downloads-v411-2744221.html`.

[38] Python Software Foundation, *Python 3.0*, Available from `https://docs.python.org/3/`.

[39] Qu R, Burke EK, McCollum B, Merlot LT and Lee SY, 2009, *A survey of search methodologies and automated system development for examination timetabling*, Journal of scheduling, **12(1)**, pp. 55–89.

[40] Schrijver A, 1998, *Theory of linear and integer programming*, John Wiley & Sons.

[41] Scientia Ltd, *Scientia timetabler*, Available from `https://www.scientia.com/`.

[42] Terashima-Marín H, Ross P and Valenzuela-Rendón M, 1999, *Evolution of constraint satisfaction strategies in examination timetabling*, Proceedings of the Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1, pp. 635–642.

[43] Thompson JM and Dowsland KA, 1998, *A robust simulated annealing based examination timetabling system*, Computers & Operations Research, **25(7)**, pp. 637–648.

[44] Tripathy A, 1984, *School timetablingâĂŤa case in large binary integer linear programming*, Management science, **30(12)**, pp. 1473–1489.

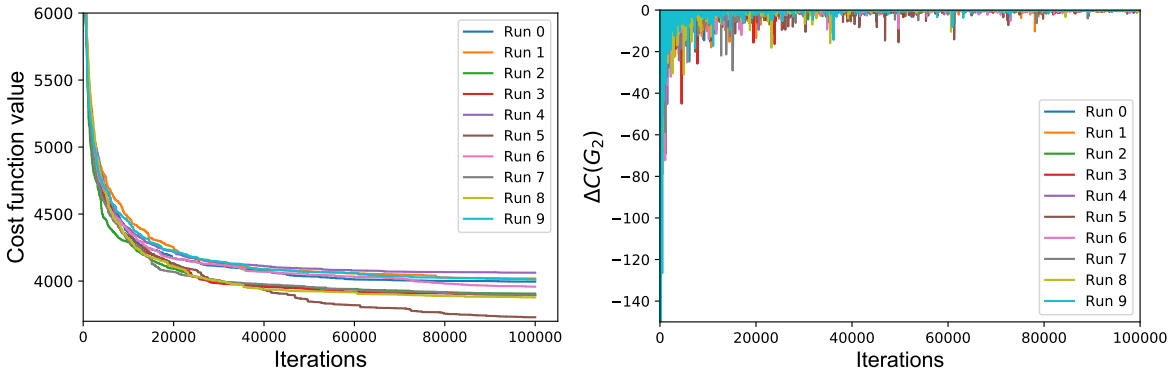
[45] Welsh DJ and Powell MB, 1967, *An upper bound for the chromatic number of a graph and its application to timetabling problems*, The Computer Journal, **10(1)**, pp. 85–86.

[46] Wood D, 1968, *A system for computing university examination timetables*, The Computer Journal, **11(1)**, pp. 41–47.

APPENDIX A

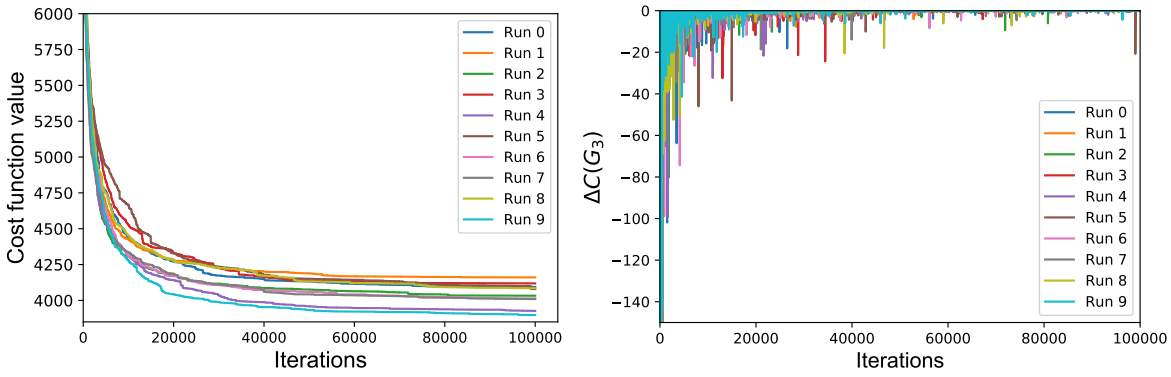
Hillclimbing parameter calibration

The graphs in this appendix are all the remaining graphs when Algorithm 3 is applied to the four timetabling graphs discussed in §4.1.4.



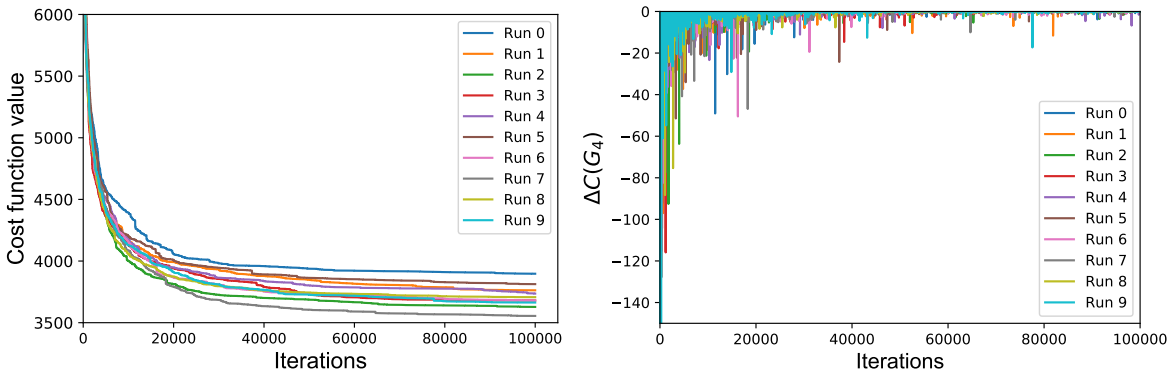
(a) (b)

FIGURE A.1: *The graphs of Algorithm 3 applied to timetabling graph G_1 using the set of rules R_2, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after 100 000 iterations and was executed 10 times on timetabling graph G_1.*



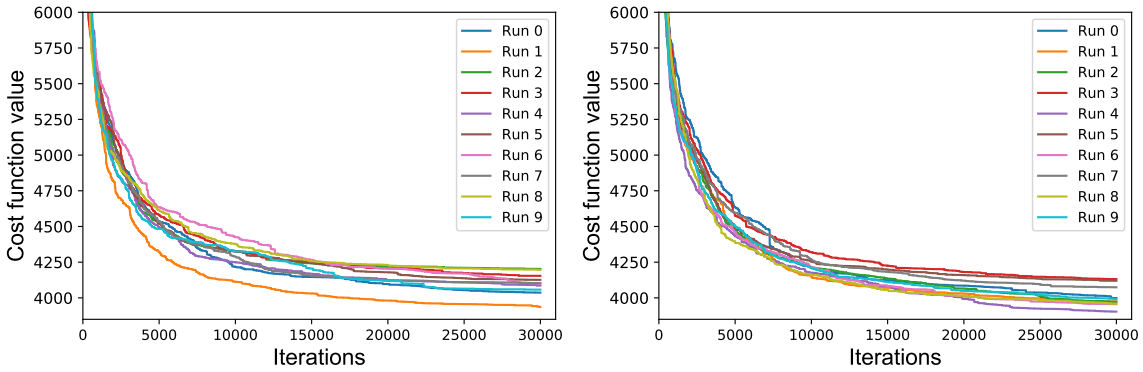
(a) (b)

FIGURE A.2: *The graphs of Algorithm 3 applied to timetabling graph G_3 using the set of rules R_2, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after 100 000 iterations and was executed 10 times on timetabling graph G_1.*

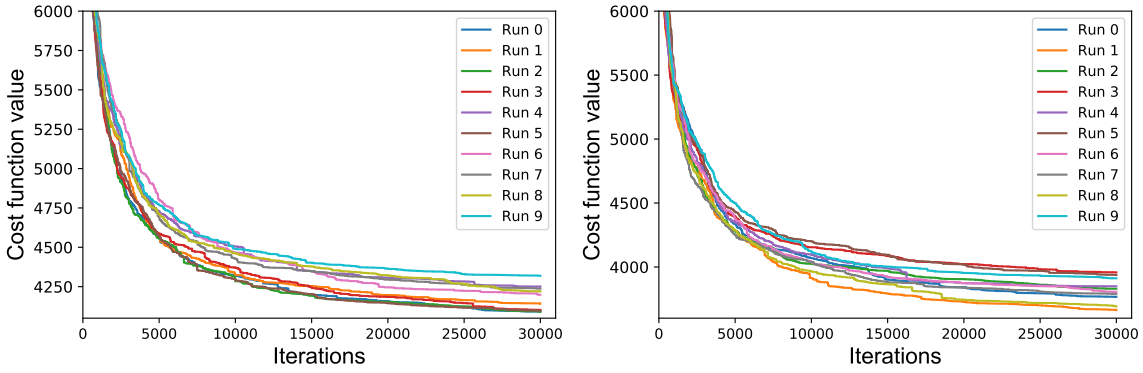


(a) (b)

FIGURE A.3: *The graphs of Algorithm 3 applied to timetabling graph G_4 using the set of rules R_2, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after 100 000 iterations and was executed 10 times on timetabling graph G_1.*



(a) G_1 (b) G_2



(c) G_3 (d) G_4

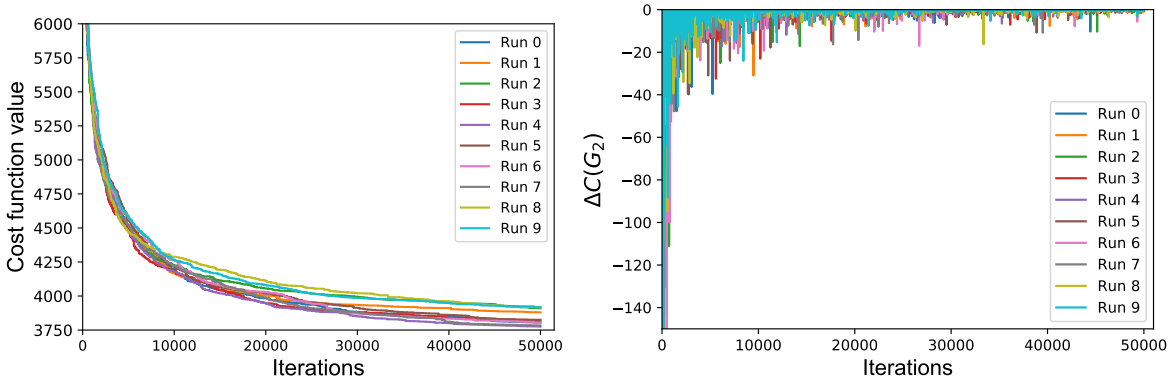
FIGURE A.4: *Graphs of the cost function value over the number of iterations when Algorithm 3 is executed on the fourth timetabling graphs obtained in §4.1.4. The parameters used are* $\gamma = 30\,000$ *and ruleset* R_3*, and again the algorithm was executed* 10 *times on each timetabling graph.*

APPENDIX B

The great deluge paramater calibration

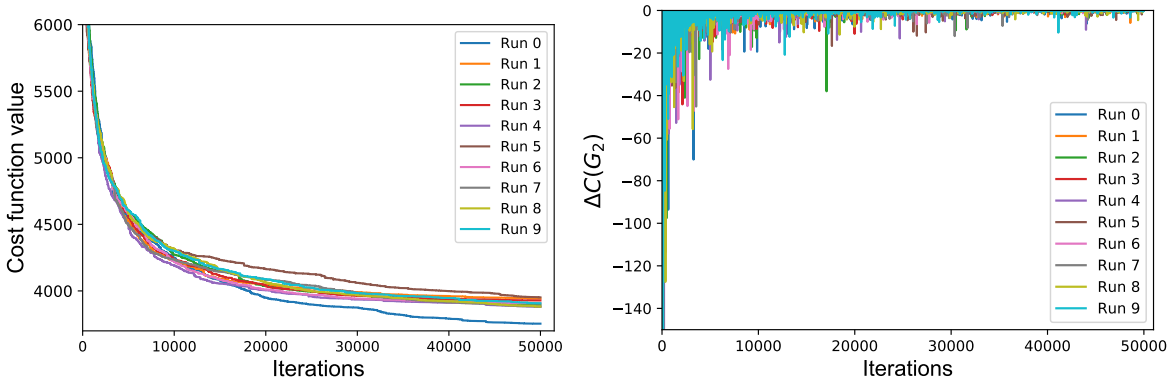
The graphs in this chapter show how the cost function changed over time when using Algorithm 4 with parameter settings $\gamma = 50\,000$ and ruleset R_3 for different values of β in

$$D(\theta) = \theta - \frac{\theta - C(neighbour\_solution)}{\beta}.$$



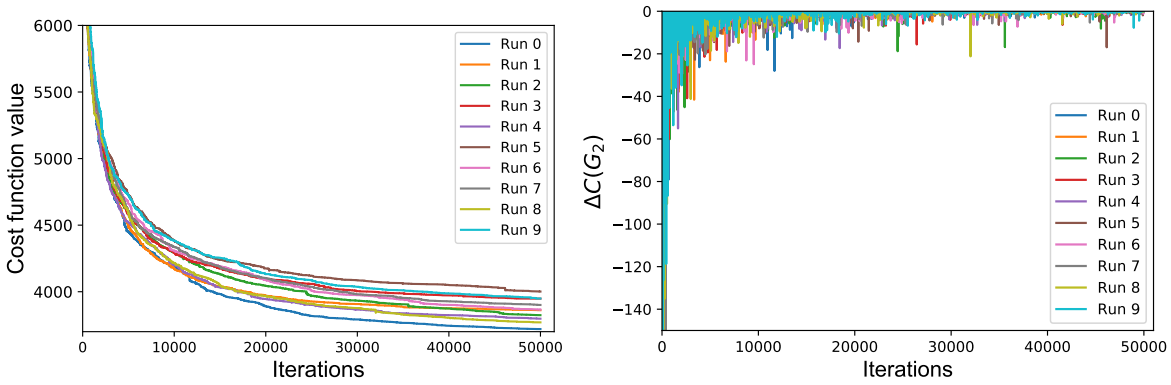
(a) (b)

FIGURE B.1: *The graphs of Algorithm 4 applied to timetabling graph G_2 using the set of rules R_3 and $\beta = 1.5$, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after 50 000 iterations and was executed 10 times on timetabling graph G_2.*



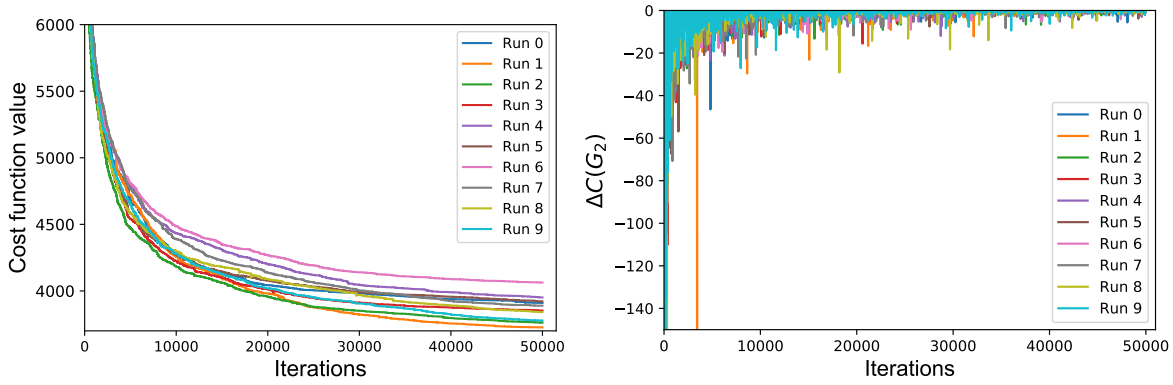
(a) (b)

FIGURE B.2: *The graphs of Algorithm 4 applied to timetabling graph G_2 using the set of rules R_3 and $\beta = 2$, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after 50 000 iterations and was executed 10 times on timetabling graph G_2.*



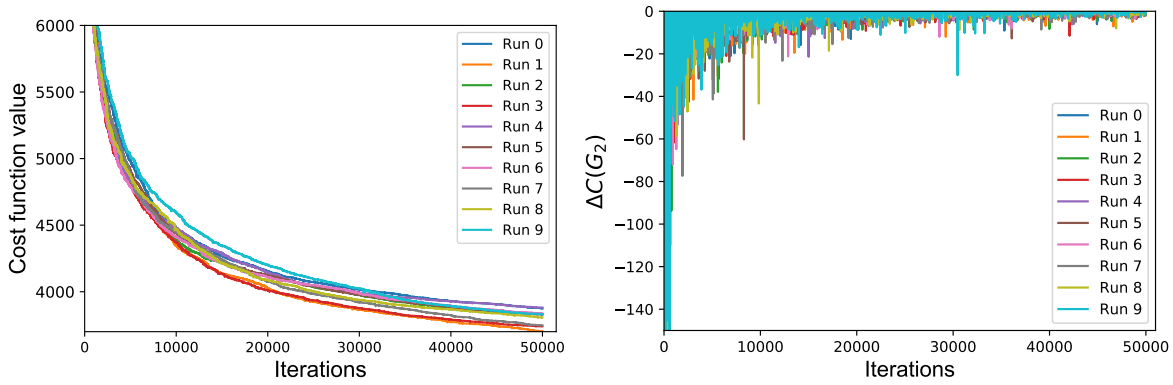
(a) (b)

FIGURE B.3: *The graphs of Algorithm 4 applied to timetabling graph G_2 using the set of rules R_3 and $\beta = 3$, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after 50 000 iterations and was executed 10 times on timetabling graph G_2.*



(a) (b)

FIGURE B.4: *The graphs of Algorithm 4 applied to timetabling graph G_2 using the set of rules R_3 and $\beta = 4$, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after 50 000 iterations and was executed 10 times on timetabling graph G_2.*



(a) (b)

FIGURE B.5: *The graphs of Algorithm 4 applied to timetabling graph G_2 using the set of rules R_3 and $\beta = 10$, where (a) is the graph of the cost function value over the iterations and (b) is the graph of the change in the cost function value at each iteration. The algorithm was set to terminate after 50 000 iterations and was executed 10 times on timetabling graph G_2.*

APPENDIX C

Cost function of the ITC

More detail on the cost function used in the 2nd International Timetabling Competition (ITC) is given in this appendix, as well as how this cost function had to be adapted to fit the variant of the ETP used in this project.

The cost function of the ITC is to minimise [30]

$$\sum_{s \in \mathcal{S}} (w_1 C_{1,s} + w_2 C_{2,s} + w_3 C_{3,s}) + w_4 C_{4,s} + w_5 C_{5,s} + w_6 C_{6,s}, \tag{C.1}$$

where the summation is over all students $s \in S$. The first term of (C.1), $C_{1,s}$, is associated with the penalty with regards to how many consecutive examination papers student s is required to write on the same day. This term receives an increment of one whenever a student s has to write two consecutive papers during the same day, where w_1 is the weight associated with $C_{1,s}$.

The variable $C_{2,s}$ will receive an increment of one whenever student s is required to write two non-consecutive examination papers within the same day, and w_2 is the weight associated with this penalty. It will be impossible for students at Stellenbosch University to write two non-consecutive examination papers on the same day, since Stellenbosch University only uses two examination timeslots per day. Thus, the term $w_2 C_{2,s}$ is not used for the purposes of this project.

The term $w_3 C_{3,s}$ is associated with the spacing of examination papers within a certain period for student s. Given an input parameter g, $C_{3,s}$ will receive an increment of one whenever student s is required to write two examination papers within g timeslots of one another. The constant w_3 is used to weight $C_{3,s}$.

The last three terms in (C.1) is not applicable to the variant of the ETP used in this project. The term $w_4 C_{4,s}$ is used to penalise examination timetables in which the larger examination papers (*i.e.* if a large number of students are enrolled for the module) are schedule late in the examination period. This is, however, not in the scope of this project. The second last term, $w_5 C_{5,s}$, is associated with the penalty applied to examination venues. Examination venues are also not in the scope of this project. Finally, the term $w_6 C_{6,s}$ is associated with a penalty to timeslots that are over-used. This is also not an issue in this project, since an equitable colouring is an requirement in this project.

This results in equation

$$\sum_{s \in \mathcal{S}} (w_1 C_{1,s} + w_3 C_{3,s}),$$

that will be used in this project. Note that the purpose of the cost function in this project is to spread out students' examination papers as much as possible. Thus, deleting terms in the cost function of the ITC that are not related to students' papers' spreads, should not influence how well the cost function used in the ITC can spread out students' examination papers.

The weights of w_1 and w_3 had to be determined for this project. These weights were obtained by inspecting all the datasets provided by the ITC, to find the one that is most similar to that of Stellenbosch University. A dataset of an university was found that also use two timeslots per day, and 607 examination papers had to be scheduled over the course of 54 timeslots [2]. The input parameters for this datasets was $w_1 = 7$ and $w_3 = 5$, and these values will also be used in this project.